Internet Downtime Prediction Web APP using Python, Machine Learning, Github and Streamlit

PROJECT REPORT

Submitted by

SURAJ RAVISHANKAR YADAV

[EC2331201010110]

Under the Guidance of

Prof. Muthuselvam

(Assistant Professor, Directorate of Online Education)

in partial fulfillment for the award of the degree of

BACHELOR OF COMPUTER APPLICATIONS
(Specialization in Data Science)

DIRECTORATE OF ONLINE EDUCATION

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

NOV/DEC 2025

DIRECTORATE OF ONLINE EDUCATION

SRM INSTITUTE OF SCIENCE AND

TECHNOLOGYKATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

This project report titled **"Internet Downtime Prediction Web APP using Python, Machine Learning, Github and Streamlit"** is the Bonafide work of **"SURAJ RAVISHANKAR YADAV [EC2331201010110]"**, who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other internship report or project report or any dissertation based on which a degree or award was conferred on an earlier occasion on this or any other candidate.

CANDIDATE DECLARATION

Project Title **Internet Downtime Prediction Web APP using Python, Machine Learning, Github and Streamlit**

I, **Suraj Ravishankar Yadav**

(Enrolment number : **EC2331201010110**), declare that this project is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this project has not previously been submitted for assessment in any academic institution, and that I have not copied in part or whole or otherwise plagiarized the work of other persons. I confirm that I have identified and declared all possible conflicts that I may have.

Signed and submitted

Date:

ACKNOWLEDGEMENTS

I express my humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support. I extend my sincere thanks to Director DOE, SRM Institute of Science and Technology, **Dr. Manoranjan Pon. Ram,** for his invaluable support. I want to convey my thanks to Programme Coordinator **Prof. Muthuselvam**, Directorate of online Education, SRM Institute of Science and Technology, for his inputs during the project reviews and support throughout the project work. Once again, my inexpressible respect and thanks to my guide, **Prof. Muthuselvam**, Assistant Professor & Programme Coordinator Directorate of online Education, SRM Institute of Science and Technology, for providing me with an opportunity to pursue my project under his mentorship. He provided me with the freedom and support to explore the research topics of my interest. His passion for solving problems and making a difference in the world has always been inspiring. I sincerely thanks to the Directorate of online Education, staff and students, SRM Institute of Science and Technology, for their help during my project. Finally, I would like to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

Suraj Ravishankar Yadav

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

df: DataFrame
pd: pandas
ML: Machine Learning
AI: Artificial Intelligence
DL: Deep Learning
OS: Operating System
LR: Logistic Regression
3D: Three-Dimensional
SQL: Structured Query Language
CSV: Comma-Separated Values
sns: seaborn
sys: System
EDA: Exploratory Data Analysis
SSD: Solid State Drive
HDD: Hard Disk Drive
RAM: Random Access Memory
LRC: Logistic Regression Classifier
DTC: Decision Tree Classifier
RFC: Random Forest Classifier
SVM: Support Vector Machine
SVC: Support Vector Classifier
KNN: K-Nearest Neighbor
GPU: Graphics Processing Unit
CPU: Central Processing Unit
UCI: University of California, Irvine.
macOS: Macintosh Operating System
IDE: Integrated Development Environment
VRAM: Video Random Access Memory
Mbps: Megabits Per Second
corr: Correlation
SciPy: Scientific Python
NumPy: Numerical Python
sklearn: scikit-learn
GIL: Global interpreter lock

1. ABSTRACT

The Internet Downtime Prediction project aims to develop a machine learning-based web application capable of predicting the severity of internet downtime for Internet Service Providers (ISPs). Maintaining consistent network connectivity is a critical challenge for ISPs, as frequent or prolonged downtimes lead to customer dissatisfaction, service complaints, and potential revenue loss. This project leverages supervised machine learning techniques to predict downtime categories - Low, Moderate, or High, based on key network performance indicators such as download speed, upload speed, latency, jitter, and packet loss, along with regional and environmental factors like city, locality, and weather conditions. The dataset was pre-processed through encoding of categorical variables and scaling of numerical features to ensure balanced input for the model. Various classification algorithms were trained and evaluated, including Logistic Regression, Decision Tree, Random Forest, Gradient Boosting, K-Nearest Neighbors, and Support Vector Machine (SVM). Among these, the **Random Forest Classifier** demonstrated the best performance with an **accuracy of 92.00%**, making it the chosen model for deployment. The trained model was serialized using the joblib library and integrated into an interactive web application built with **Streamlit**, allowing users to input network and environmental parameters to predict the expected downtime category. This system enables ISPs to proactively monitor network conditions, identify high-risk zones, and take preventive measures to reduce severe service disruptions. In conclusion, the Internet Downtime Prediction project demonstrates the effective application of machine learning in network reliability management, offering a practical tool for improving service quality and customer experience.

2. INTRODUCTION

2.1 Brief Overview of the Project

The Internet Downtime Prediction project aims to develop a machine learning model to predict the severity of internet downtime for ISPs. Stable connectivity is vital to avoid customer dissatisfaction and revenue loss. The project uses network performance indicators, such as download/upload speed, latency, jitter, and packet loss, along with regional and environmental factors like city, locality, and weather, to classify downtime as Low, Moderate, or High. The final model is deployed as an interactive Streamlit web application for real-time prediction.

2.2 Problem Statement

ISPs face challenges in maintaining uninterrupted connectivity due to fluctuations in network parameters. Accurate prediction of downtime categories enables ISPs to take proactive measures, minimize service disruptions, and improve overall reliability.

2.3 Dataset Used

The dataset contains network performance and environmental attributes, including Download and Upload Speed, Latency, Jitter, Packet Loss, Weather, City, and Locality. The target variable is Downtime Category (Low, Moderate, High). Data preprocessing involved handling missing values, encoding categorical features, scaling numerical features, and removing outliers to ensure quality and model performance.

2.4 Goal of the Project

The goal is to build a predictive model for classifying internet downtime severity. Multiple algorithms—Logistic Regression, Decision Tree, Random Forest, Gradient Boosting, KNN, and SVM—were trained and compared. The Random Forest Classifier achieved the highest accuracy (92.00%) and was deployed via a Streamlit web app, enabling ISPs to anticipate network failures and enhance service quality.

3. SYSTEM ANALYSIS

3.1 Current System

ISPs currently monitor internet downtime through **manual observation and reactive troubleshooting**. Engineers detect issues after customer complaints or monitoring reports, analyzing network logs, bandwidth, and signal data. This approach is **time-consuming**, relies on human expertise, and often results in **delayed responses**, reducing service reliability.

3.2 Limitations of Current System

The current system has several limitations, including:

- **Reactive approach:** Downtime is addressed only after it occurs.
- **High operational cost:** Continuous monitoring requires significant resources.
- **No predictive capability:** Potential downtimes cannot be forecasted.
- **Data complexity:** Manual analysis of large datasets is inefficient.
- **Inconsistent results:** Human interpretation can vary, leading to unreliable outcomes.

3.3 Proposed System

The project proposes an **automated Internet Downtime Prediction System** using machine learning. It predicts downtime severity (**Low, Moderate, High**) based on network indicators and environmental factors such as download/upload speed, latency, jitter, packet loss, and weather. This enables ISPs to **proactively prevent service interruptions** and improve reliability.

3.4 Features of Proposed System

The proposed system offers the following key features:

- **Automated Data Analysis:** Predicts downtime categories from network and environmental data.
- **Machine Learning Models:** Uses Logistic Regression, Decision Tree, Random Forest, Gradient Boosting, KNN, and SVM for robust predictions.
- **High Accuracy:** Random Forest achieved **92.00% accuracy**.
- **Real-time Prediction:** Streamlit web app allows instant input and prediction.
- **Proactive Network Management:** Helps ISPs take preventive actions before downtime.
- **Scalability & Consistency:** Can be scaled across regions and provides consistent, unbiased results.

4. SYSTEM CONFIGURATIONS

4.1 Software Requirements

- Operating system: Windows 10 Home or Above version
- Programming Language: Python
- Data Analysis and Manipulation Libraries: Numpy and Pandas
- Data Visualization Libraries: Matplotlib and Seaborn
- Machine Learning Libraries: Scikit-learn
- **Model Serialization Library:** Joblib
- Development Environment: Jupyter Notebook or Google Colab
- Spreadsheet Software: Microsoft Excel or Google Sheets
- **Text Editor:** Notepad++ or Visual Studio Code
- Version Control: Git and GitHub
- **Python Framework for Deployment:** Streamlit

4.2 Hardware Requirements

Desktop or Laptop: Any desktop or laptop with following specifications:

- Processor: Intel Core i3 or equivalent
- RAM: 4 GB (minimum)
- Hard Disk: 128 GB HDD or SSD (minimum)
- Graphics Card: Integrated graphics or dedicated graphics card with at least 1 GB of VRAM
- Monitor: 15" with at least 1366x768 resolution

4.3 Internet Connection

Broadband internet connection with at least 10 Mbps download and 5 Mbps upload speeds

5. TECHNOLOGY

5.1 Python

Figure 5.1 : Python Icon

Python is a high-level, interpreted programming language known for its simplicity and readability. It follows a design philosophy that emphasizes clear and concise code, making it suitable for both small and large-scale applications. Python supports multiple programming paradigms, such as object-oriented and functional programming, and includes a rich standard library. It is widely used in fields like data science, web development, automation, and scientific computing.

5.2 NumPy

Figure 5.2 : NumPy Icon

NumPy is a core Python library for numerical and scientific computing. It provides support for large, multi-dimensional arrays and matrices, along with a wide range of high-performance mathematical functions to operate on them. NumPy arrays are homogeneous, meaning all elements are of the same data type, which makes computations faster and memory usage more efficient. It serves as the foundation for many other scientific and data analysis libraries in Python.

5.3 Pandas

Figure 5.3 : Pandas Icon

Pandas is a powerful Python library used for data manipulation and analysis. It offers efficient data structures, such as Series and DataFrame, for handling and processing large datasets. Pandas provides a wide range of functions for data cleaning, merging, filtering, aggregation, and visualization. It is an essential tool in data science, enabling users to prepare and analyze data effectively.

5.4 Matplotlib

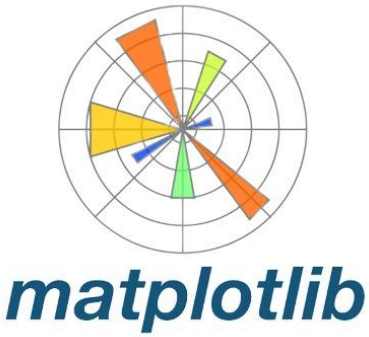

Figure 5.4 : Matplotlib Icon - Plotting Library

Matplotlib is a widely used Python library for data visualization. It provides a comprehensive set of tools for creating high-quality plots, charts, and graphs such as line plots, scatter plots, bar charts, and histograms. Matplotlib allows extensive customization of visual elements, including titles, labels, colors, and styles. Built on top of NumPy, it integrates seamlessly with other scientific libraries like Pandas and SciPy, making it an essential tool for visual data analysis.

5.5 Seaborn

Figure 5.5 : Seaborn Icon - Data Visualization Library

Seaborn is a Python library for data visualization, built on top of Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics. Seaborn is designed to work efficiently with Pandas DataFrames and simplifies the process of visualizing complex datasets. It offers a variety of plot types, including line plots, scatter plots, bar charts, histograms, and heatmaps, making it a powerful tool for exploratory data analysis.

5.6 Scikit-learn

Figure 5.6 : Scikit-learn (sklearn) Icon

Scikit-learn is a popular Python library used for machine learning. It offers a wide range of tools for data preprocessing, feature selection, model building, and evaluation. Designed to work seamlessly with NumPy and Pandas, Scikit-learn provides efficient implementations of various algorithms for classification, regression, clustering, and dimensionality reduction. Being open-source, it has a large and active community that continuously contributes to its development and enhancement.

5.7 Jupyter Notebook

Figure 5.7 : Jupyter Notebook Application Icon

Jupyter Notebook is an open-source web-based application used for interactive data analysis, scientific computing, and visualization. It enables users to create and share documents that combine live code, equations, visualizations, and explanatory text. Supporting multiple programming languages such as Python, R, and Julia, Jupyter Notebook provides an interactive environment that facilitates experimentation, learning, and collaboration. It is an essential tool in the modern data science workflow.

5.8 Joblib

Figure 5.8 : Joblib Icon - Serialization Library

Joblib is a Python library that provides efficient tools for object serialization, parallel computing, and function pipelining. It is commonly used in machine learning workflows to save and load trained models, as well as to handle large NumPy arrays efficiently. Joblib also supports parallel processing, enabling faster execution of computationally intensive tasks such as model training and evaluation.

5.9 Streamlit

Figure 5.9 : Streamlit Icon - Web Application Framework

Streamlit is an open-source Python library used to build interactive web applications for data science and machine learning. It enables users to create and deploy data-driven web apps quickly with minimal coding effort. Streamlit simplifies the process of transforming data analysis scripts into shareable, interactive dashboards, making it a valuable tool for data visualization and model deployment.

5.10 GitHub

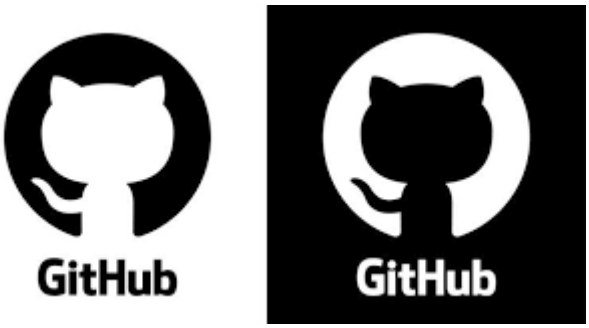

Figure 5.10 : GitHub Icon - Version Control Platform

GitHub is a web-based platform used for version control and collaborative software development. It enables developers to host, manage, and review code while maintaining version history using Git. GitHub also facilitates teamwork through features like issue tracking, pull requests, and project management tools, making it an essential platform for open-source and collaborative projects.

5.11 Notepad++

Figure 5.11 : Notepad++ Icon

Notepad++ is a free, open-source text and source code editor for Windows. It provides advanced features such as syntax highlighting, code folding, auto-completion, and regular expression search and replace. Notepad++ also supports multiple tabs, allowing users to work on several files simultaneously, making it a preferred choice for developers and programmers.

5.12 Microsoft Excel

Figure 5.12 : Microsoft Excel Icon

Microsoft Excel is a widely used spreadsheet application designed for data organization, analysis, and visualization. It enables users to manage and analyze data through tables, charts, and graphs while offering a broad range of features such as formulas, functions, pivot tables, and conditional formatting. Excel is a powerful tool for performing complex calculations, generating reports, and visualizing large datasets, making it an essential application for data analysis and business decision-making.

5.13 Anaconda Navigator

Figure 5.13 : Anaconda Navigator Icon

Anaconda Navigator is a graphical user interface included with the Anaconda distribution of Python and R. It is widely used in data science and machine learning for managing packages, environments, and applications. Anaconda Navigator provides an easy-to-use interface for launching tools such as Jupyter Notebook, Spyder, and RStudio, along with managing libraries and dependencies. It simplifies environment creation, package installation, and version control, making it an essential platform for organizing and executing data science workflows.

6. ML ALGORITHMS

6.1 Logistic Regression

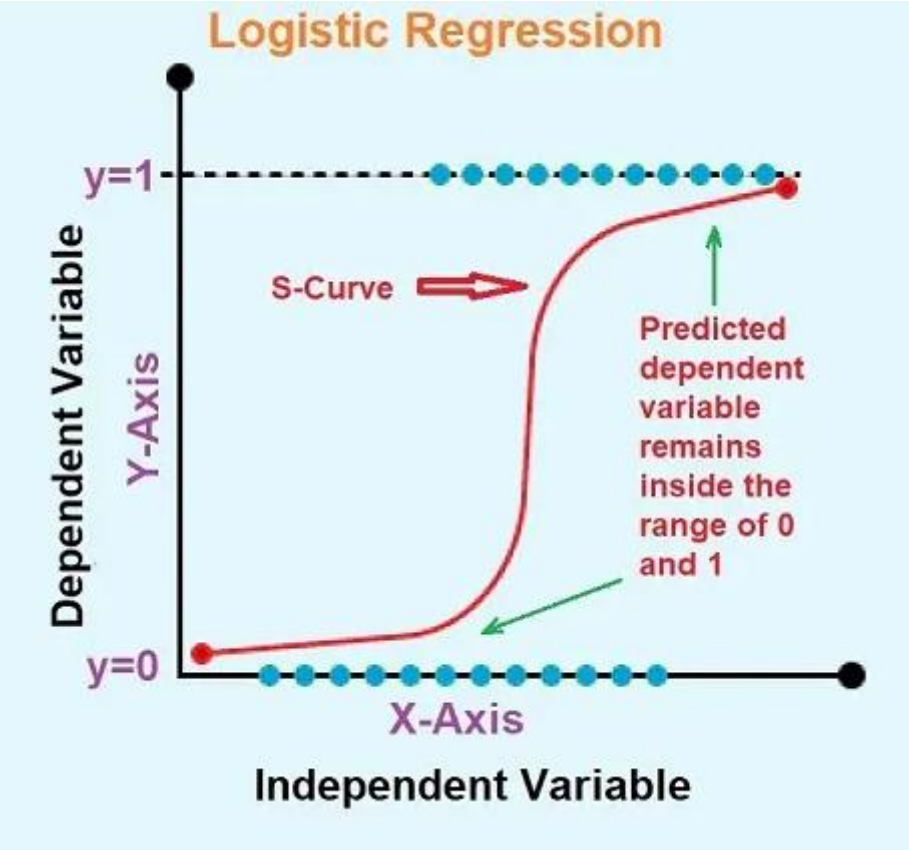

Figure 6.1 : Logistic Regression

- Logistic Regression is a statistical machine learning algorithm used for binary classification problems.
- It models the probability of an output variable (such as “yes” or “no”) based on one or more input variables.
- The algorithm fits a logistic (sigmoid) curve to the data, mapping the input variables to the predicted probability of the target class.
- It is widely used in various domains such as healthcare, finance, and marketing for predicting the likelihood of an event occurring.
- Logistic Regression provides a simple and interpretable way to model the probability of a binary outcome and often serves as a baseline model for more complex machine learning algorithms.

6.2 Decision Tree Classifier

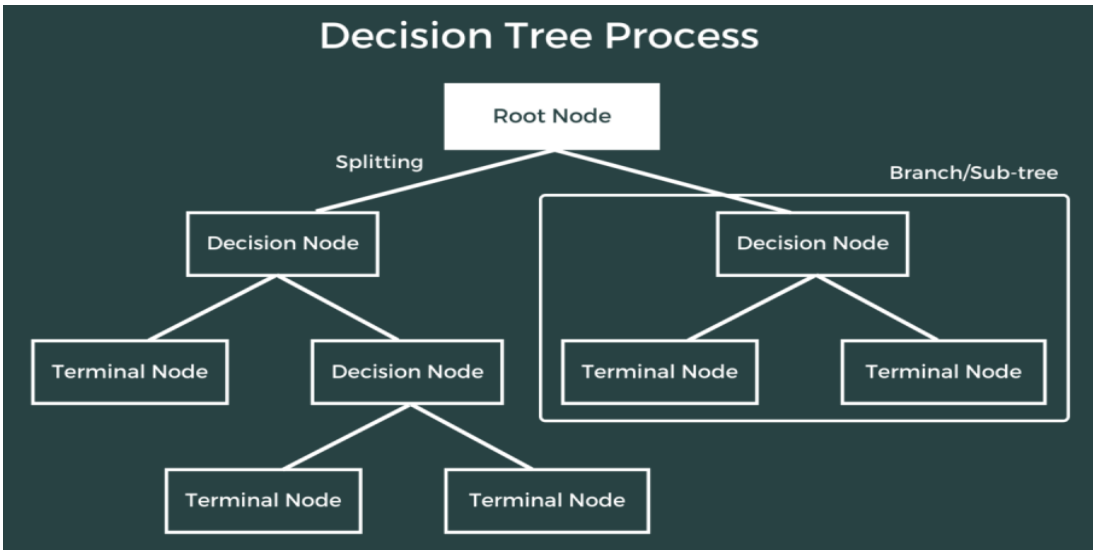

Figure 6.2 : Decision Tree Classifier

- Decision Tree Classifier is a popular machine learning algorithm used for solving classification problems.
- It is a type of supervised learning algorithm that builds a tree-like model of decisions and their possible outcomes.
- The algorithm works by recursively splitting the dataset into subsets based on the most significant feature, forming a tree-like structure.
- At each split, the feature that provides the highest information gain—which measures the reduction in entropy or impurity—is selected.
- Decision Trees are widely used in various domains such as healthcare, finance, and marketing for predicting the likelihood of specific outcomes.
- They provide a simple, interpretable, and visual way to understand the relationship between input features and the target variable.
- Decision Trees are also commonly used as baseline models and form the foundation for more advanced ensemble algorithms like Random Forest and Gradient Boosting.

6.3 Random Forest Classifier

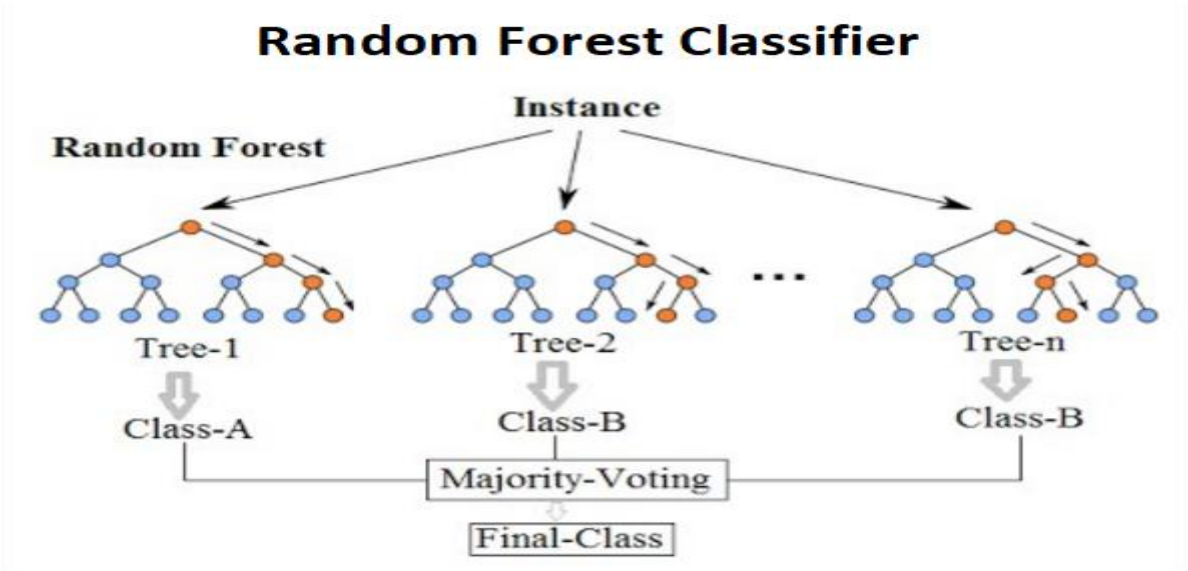

Figure 6.3 : Random Forest Classifier

- Random Forest Classifier is a powerful ensemble learning algorithm used for classification problems.
- It is a supervised learning method that builds multiple decision trees and combines their outputs to make a final prediction.
- The algorithm works by randomly selecting subsets of data samples and features from the training dataset to build each decision tree.
- Each tree makes an independent prediction, and the final output is determined by majority voting among all trees.
- Random Forest reduces overfitting and improves accuracy compared to a single decision tree.
- It can efficiently handle large datasets with high-dimensional features.
- The algorithm is widely used across various domains such as healthcare, finance, and marketing for predicting the likelihood of specific outcomes.
- Random Forest provides a robust, interpretable, and scalable solution for both classification and regression tasks.

6.4 Gradient Boosting Classifier

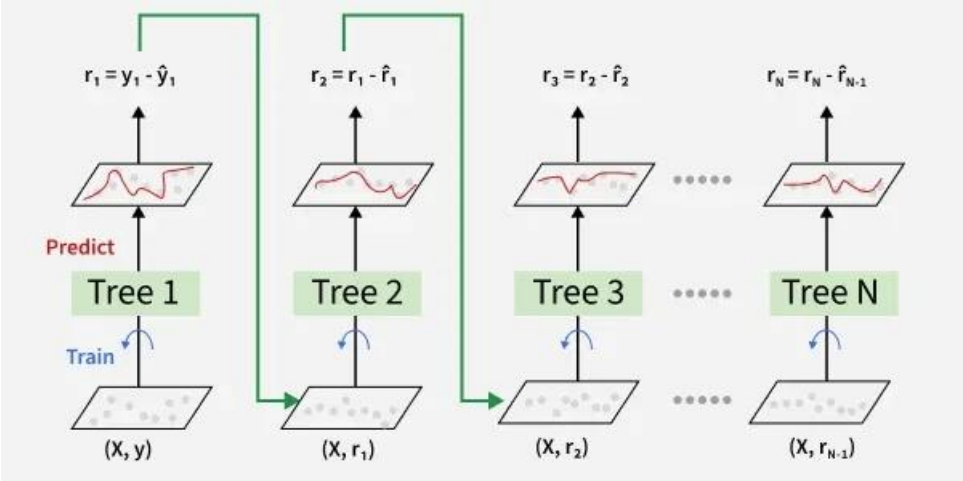

Figure 6.4 : Gradient Boosting Classifier

- Gradient Boosting Classifier is an advanced ensemble machine learning algorithm used primarily for classification tasks.
- It is a supervised learning method that builds multiple weak learners, usually decision trees, in a sequential manner.
- Each new tree is trained to correct the errors made by the previous trees, gradually improving the model's overall accuracy.
- The algorithm minimizes a loss function (such as log loss or mean squared error) by using gradient descent optimization to find the best model parameters.
- Gradient Boosting combines the predictions of all weak learners to produce a strong predictive model.
- It is highly effective in capturing complex patterns and relationships within data.
- The algorithm is less prone to overfitting compared to individual models, especially when parameters like learning rate, number of estimators, and tree depth are properly tuned.
- Gradient Boosting is widely used in various fields such as finance, healthcare, marketing, and predictive analytics for tasks like classification, risk assessment, and forecasting.
- It provides high accuracy, flexibility, and robustness, making it one of the most powerful algorithms in modern machine learning.

6.5 K-Nearest Neighbors Classifier

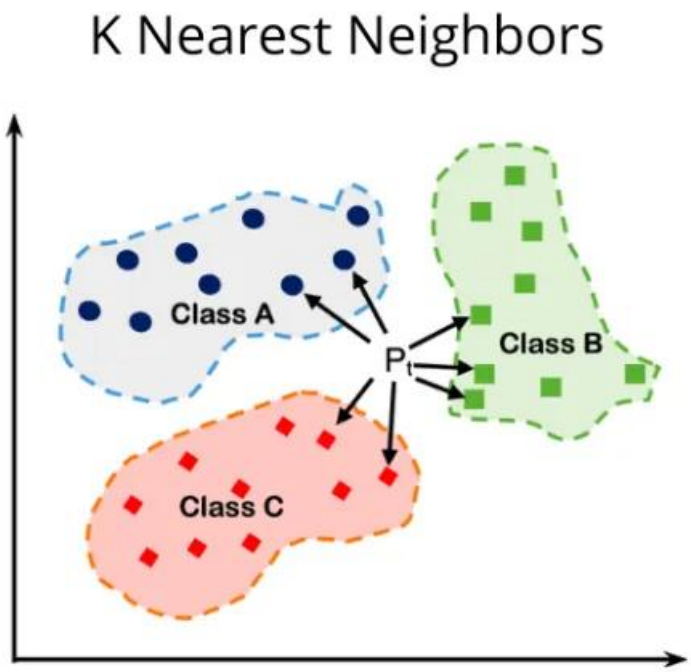

Figure 6.5 : K-Nearest Neighbors Classifier

- K-Nearest Neighbors (KNN) Classifier is a simple yet powerful supervised machine learning algorithm used for classification and regression tasks.
- It predicts the class of a data point based on the majority class of its 'k' nearest neighbors.
- The choice of 'k' affects performance: smaller values are sensitive to noise, larger values smooth boundaries.
- Distance metrics like Euclidean, Manhattan, or Minkowski measure similarity between points.
- KNN is instance-based, storing the training dataset and predicting only for new inputs.
- It is widely used in pattern recognition, recommendation systems, and image classification.
- Performs well on small to medium datasets, though computationally intensive for large datasets. It serves as a baseline algorithm for many classification tasks and remains an essential part of the machine learning toolkit.

6.6 Support Vector Classifier (SVC)

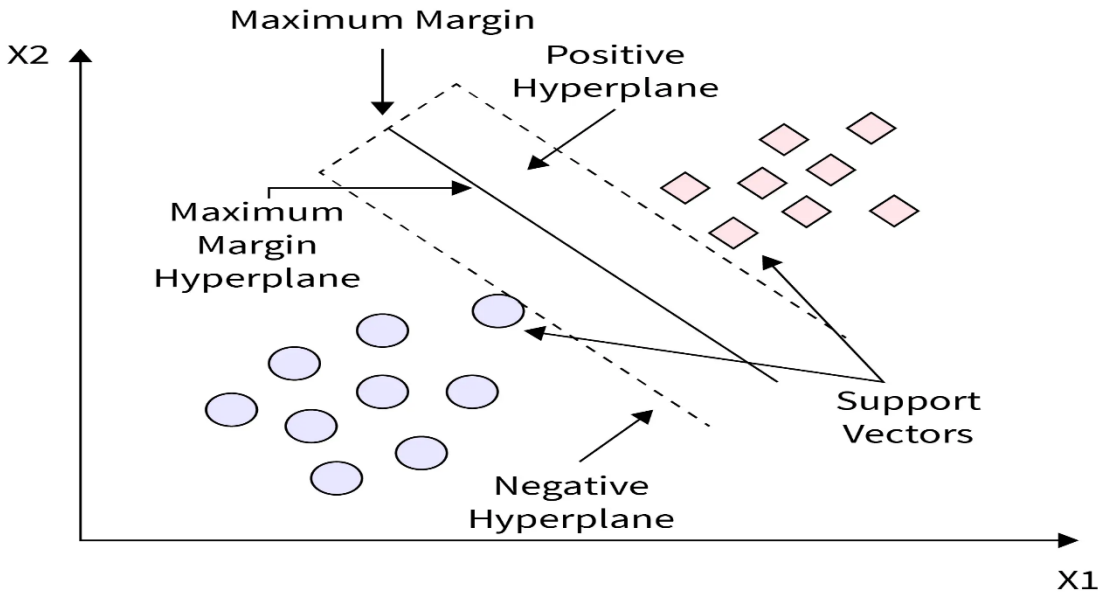

Figure 6.6 : Support Vector Classifier

- The Support Vector Classifier (SVC) is a supervised machine learning algorithm used for classification tasks.
- It works by finding the optimal hyperplane that best separates data points of different classes in a feature space.
- SVC focuses on maximizing the margin — the distance between the nearest data points (called support vectors) and the separating hyperplane.
- It can handle both linear and non-linear classification using different kernel functions such as linear, polynomial, and radial basis function (RBF).
- SVC performs well for high-dimensional data and provides good accuracy even when the number of features is greater than the number of samples.
- It is sensitive to feature scaling, so standardization or normalization of data is recommended before training.
- However, SVC can be computationally expensive for large datasets.

7. CODING

7.1 Internet_Downtime_Prediction.ipynb file

```
# Suppressing display of warnings
import warnings
warnings.filterwarnings('ignore')

# Import libraries and modules

# 1. System and Core Libraries
import sys
import numpy as np
import pandas as pd

# 2. Visualization Libraries
import matplotlib.pyplot as plt
import seaborn as sns

# 3. Machine Learning - Model Building
import sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC


# 4. Preprocessing and Model Evaluation
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# 5. Model Saving
import joblib

# Checking library versions
```

```
print('Python: {}'.format(sys.version))
print('numpy: {}'.format(np.__version__))
print('pandas: {}'.format(pd.__version__))
print('matplotlib: {}'.format(plt.matplotlib.__version__))
print('seaborn: {}'.format(sns.__version__))
print('sklearn: {}'.format(sklearn.__version__))

# Loading the dataset
df = pd.read_csv('ISP_6months_data.csv')
df

# Checking the dimensions of dataset
df.shape

# Checking columns of dataset
df.columns

# Check for missing values
df.isnull().sum()

# Check for duplicate value
df.duplicated().sum()

# Display information about dataset
df.info()

# Statistical Summary
df.describe()

# Computing pairwise correlations
df.select_dtypes(include=['number']).corr()

# Data Extraction

## Extracting Date-Time Features

# Extracting Hour, Day, and Month columns from the Timestamp column
df['Timestamp'] = pd.to_datetime(df['Timestamp'])
df['Hour'] = df['Timestamp'].dt.hour
df['Day'] = df['Timestamp'].dt.day
```

```
df['Month'] = df['Timestamp'].dt.month
df['Year'] = df['Timestamp'].dt.year

# Dropping the Timestamp column since the required data has been extracted
df = df.drop('Timestamp', axis=1)

# Display the updated DataFrame
df.head()

## Dropping Unnecessary Columns

# Dropping the Downtime_min column since Downtime_Category already
represents the same information
df = df.drop('Downtime_min', axis=1)

# Display the updated DataFrame
df.head()

# Data Visualization

# Visualizing countplot on target variable

sns.countplot(x='Downtime_Category', data=df, palette='Set1')
plt.title('Distribution of Downtime Categories')
plt.xlabel('Downtime Category')
plt.ylabel('Count')
plt.show()


# Checking the distribution of each variable
df.hist(bins=10, figsize=(14,10))
plt.show()

# Define the columns to plot
columns = ['DownloadSpeed_Mbps', 'UploadSpeed_Mbps',
           'Latency_ms', 'Jitter_ms', 'PacketLoss_%']

# Create the subplots
fig, ax = plt.subplots(ncols=5, figsize=(16, 4))
fig.suptitle('Distribution Plots')
```

```
# Create the distribution plots
for index, column in enumerate(columns):
    sns.distplot(df[column], ax=ax[index])
    ax[index].set_title(column)

# Show the plot
plt.show()

# box plot
num_features = ['DownloadSpeed_Mbps', 'UploadSpeed_Mbps',
               'Latency_ms', 'Jitter_ms', 'PacketLoss_%']
plt.figure(figsize=(12, 8))
for i, col in enumerate(num_features, 1):
    plt.subplot(3, 2, i)
    sns.boxplot(x=df[col], color='skyblue')
    plt.title(f'Boxplot of {col}')
plt.tight_layout()
plt.show()

# Visualizing pairplot
sns.pairplot(data = df, hue = 'Downtime_Category')
plt.show()

# Correlation Heatmap
plt.figure(figsize=(12,8))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm',
fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()

## Handling Outliers

numeric_cols = ['DownloadSpeed_Mbps', 'UploadSpeed_Mbps', 'Latency_ms',
               'Jitter_ms', 'PacketLoss_%', 'Complaints',]

for col in numeric_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
```

```
    df = df[(df[col] >= Q1 - 1.5 * IQR) & (df[col] <= Q3 + 1.5 * IQR)]

df

# Features and target
X = df.drop(['Hour','Day', 'Month', 'Year', 'Downtime_Category'], axis=1)
y = df['Downtime_Category']
print(X.head())
print(y.head())

# Identify categorical and numerical columns
cat_cols = X.select_dtypes(include=['object']).columns
num_cols = X.select_dtypes(exclude=['object']).columns
cat_cols, num_cols

# Define preprocessor
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(handle_unknown='ignore'), cat_cols),
        ('num', StandardScaler(), num_cols)
    ]
)
preprocessor

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
X_train.head(2), X_test.head(2), y_train.head(2), y_test.head(2)

# Define models
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Support Vector Machine': SVC()
}

# Evaluate models
```

```
results = {}
for name, clf in models.items():
    pipe = Pipeline(steps=[('preprocess', preprocessor), ('classifier', clf)])
    pipe.fit(X_train, y_train)
    y_pred = pipe.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    results[name] = acc
    print(f"{name}: {acc * 100:.2f}%")

# Convert results dictionary to DataFrame
acc_df = pd.DataFrame(list(results.items()), columns=['Model', 'Accuracy'])

# Multiply by 100 for percentages
acc_df['Accuracy'] = acc_df['Accuracy'] * 100

# Seaborn barplot
plt.figure(figsize=(8, 5))
sns.barplot(x='Accuracy', y='Model', data=acc_df, palette='Set2')
plt.xlabel('Accuracy (%)')
plt.ylabel('Model')
plt.title('Model Accuracies')
plt.xlim(0, 100)  # Because we multiplied by 100
plt.show()


# finding best model
best_model_name = max(results, key=results.get)
print(f"Best Model: {best_model_name} with accuracy
{results[best_model_name]*100:.2f}%")

#saving the best model
best_clf = models[best_model_name]
best_pipeline = Pipeline(steps=[('preprocess', preprocessor), ('classifier',
best_clf)])
best_pipeline.fit(X_train, y_train)
joblib.dump(best_pipeline, f"{best_model_name.replace(' ',
'_')}_model.joblib")

# Testing on sample data
sample_data_1 = ['Mumbai', 'Bandra', 'Sunny', 50.3532, 38.3903, 4.2588,
```

```
1.2102, 0.0597, 0.6759]
sample_data_2 = ['Chennai', 'T Nagar', 'Cloudy', 43.9800, 10.4696, 26.0310,
                                                    6.2253,   1.2262,
                                                    11.5179]
sample_data_3 = ['Bangalore', 'Whitefield', 'Stormy', 4.0357, 1.0286, 99.7312,
27.6730,                                            5.3632,
                                                    63.1063]

# Combine into a list of lists (for multiple predictions)
samples = [sample_data_1, sample_data_2, sample_data_3]

# Convert to DataFrame with column names
sample_df = pd.DataFrame(samples, columns=['City', 'Locality',
'WeatherCondition',
                        'DownloadSpeed_Mbps', 'UploadSpeed_Mbps',
                        'Latency_ms', 'Jitter_ms', 'PacketLoss_%',
'Complaints'])

# Predict using the trained pipeline
predictions = best_pipeline.predict(sample_df)

# Display results
for i, pred in enumerate(predictions, 1):
    print(f"Predicted Downtime Category for Sample {i}: {pred}")


# Load the saved Random Forest model
loaded_model = joblib.load('Random_Forest_model.joblib')

# Make predictions
predictions = loaded_model.predict(sample_df)

# Display results
for i, pred in enumerate(predictions, 1):
    print(f"Predicted Downtime Category for Sample {i}: {pred}")
```

7.2 Streamlit_App.py file

```
import numpy as np
import pandas as pd
import joblib
import streamlit as st
from PIL import Image

# Load model
model = joblib.load('Random_Forest_model.joblib')

# Display header image
image = Image.open('internet.png')
st.image(image.resize((1000, 300)))

def internet_downtime_prediction(City, Locality, WeatherCondition,
DownloadSpeed_Mbps, UploadSpeed_Mbps, Latency_ms, Jitter_ms,
PacketLoss, Complaints):
    features = pd.DataFrame([{
        'City': City,
        'Locality': Locality,
        'WeatherCondition': WeatherCondition,
        'DownloadSpeed_Mbps': DownloadSpeed_Mbps,
        'UploadSpeed_Mbps': UploadSpeed_Mbps,
        'Latency_ms': Latency_ms,
        'Jitter_ms': Jitter_ms,
        'PacketLoss_%': PacketLoss,
        'Complaints': Complaints
    }])
    prediction = model.predict(features)
    return prediction[0]

# Main app
def main():
    st.title(" 🌐 Internet Downtime Prediction")

    # City to Locality mapping
    city_localities = {
        "Mumbai": ["Andheri", "Bandra", "Dadar", "Colaba", "Kurla"],
```

```
        "Chennai": ["T Nagar", "Adyar", "Tambaram", "Velachery", "Anna Nagar"],
        "Delhi": ["Dwarka", "Rohini", "Saket", "Lajpat Nagar", "Karol Bagh"],
        "Bangalore": ["Indiranagar", "Koramangala", "Whitefield", "HSR Layout",
"BTM"],
        "Pune": ["Baner", "Hinjewadi", "Kothrud", "Viman Nagar", "Wakad"]
    }

    # City selection
    City = st.selectbox("City", list(city_localities.keys()), index=None,
placeholder="Select a City")

    # Locality selection (depends on City)
    if City:
        Locality = st.selectbox("Locality", city_localities[City], index=None,
placeholder="Select a Locality")
    else:
        Locality = st.selectbox("Locality", [], index=None, placeholder="Select a
City first")

    WeatherCondition = st.selectbox("Weather Condition", ['Sunny', 'Stormy',
'Cloudy', 'Rainy'], index=None, placeholder="Select Weather Condition")

    DownloadSpeed_Mbps = st.number_input(
        "Download Speed (Mbps) [Range: 0.0 – 200]",
        min_value=0.0, max_value=200.0,
        format="%.1f",   # Limits input to 1 decimal places
        step=0.1       # Increases/decreases by 0.1
    )

    UploadSpeed_Mbps = st.number_input(
        "Upload Speed (Mbps) [Range: 0.0 – 50.0]",
        min_value=0.0, max_value=50.0,
        format="%.1f",   # Limits input to 1 decimal places
        step=0.1       # Increases/decreases by 0.1
    )

    Latency_ms = st.number_input(
        "Latency (ms) [Range: 0.5 – 145.0]",
        min_value=0.5, max_value=145.0,
```

```
        format="%.1f",   # Limits input to 1 decimal places
        step=0.1       # Increases/decreases by 0.1
    )

    Jitter_ms = st.number_input(
        "Jitter (ms) [Range: 0.1 – 35.0]",
        min_value=0.1, max_value=35.0,
        format="%.1f",   # Limits input to 1 decimal places
        step=0.1       # Increases/decreases by 0.1
    )

    PacketLoss = st.number_input(
        "Packet Loss (%) [Range: 0.0 – 7.0]",
        min_value=0.0, max_value=7.0,
        format="%.1f",   # Limits input to 1 decimal places
        step=0.1       # Increases/decreases by 0.1
    )

    Complaints = st.number_input(
        "Complaints [Range: 0 – 100]",
        min_value=0, max_value=100,
        format="%d",   # Limits input to 0 decimal places
        step=1       # Increases/decreases by 1
    )

    if st.button("Predict Downtime"):
        result = internet_downtime_prediction(
            City, Locality, WeatherCondition,
            DownloadSpeed_Mbps, UploadSpeed_Mbps,
            Latency_ms, Jitter_ms, PacketLoss, Complaints
        )

        # Default color for label
        label_html = f'<span style="color: red; font-weight:bold;">Predicted
Downtime Category:</span> '

        # Set color based on prediction
        if result == "Low_Downtime":
            value_html = f'<span style="color: lightgreen; font-
```

```
weight:bold;">{result}</span>'
        elif result == "Moderate_Downtime":
            value_html = f'<span style="color: yellow; font-
weight:bold;">{result}</span>'
        elif result == "High_Downtime":
            value_html = f'<span style="color: orange; font-
weight:bold;">{result}</span>'
        else:
            value_html = f'<span style="color: black;">{result}</span>'

        # Display colored prediction
        st.markdown(label_html + value_html, unsafe_allow_html=True)

        # Show About info after prediction
        st.info(
            """
            **Classifier:** Random Forest Classifier
            **Accuracy:** 92.00 %
            **Built by:** Suraj R. Yadav
            """
        )

if __name__ == '__main__':
    main()
```

7.3 Requirements.txt file

```
numpy==1.26.4

pandas==2.2.2

matplotlib==3.9.0

seaborn==0.13.2

scikit-learn==1.4.2

joblib==1.4.2

streamlit==1.35.0
```

8. IMPLEMENTATION

8.1 Data Understanding

8.1.1 Dataset Information

The dataset used for this project contains real-world network performance metrics collected from multiple cities and localities to analyze and predict internet downtime severity. The data includes various numerical and categorical features related to internet performance and environmental conditions.

Dataset Type	Multivariate
Subject Area	Data Science / Network Analytics
Associated Tasks	Classification
Feature Type	Mixed (Numerical and Categorical)
# Instances (Rows)	10,000 (approx.)
# Features (Columns)	9
Target Variable	Downtime Category (Low, Moderate, High)

8.1.2 Additional Dataset Information

The dataset combines technical network metrics and environmental attributes to help predict service interruptions. The network-related features include *Download Speed*, *Upload Speed*, *Latency*, *Jitter*, and *Packet Loss*, while external features such as *Weather Condition*, *City*, and *Locality* contribute to understanding the influence of environmental and geographical factors.

8.1.3 Variables Table

Variable Name	Role	Type	Description
Timestamp	Feature	Categorical	Represents the date and time when the network performance data was recorded.
City	Feature	Categorical	Indicates the city where the network data was collected.
Locality	Feature	Categorical	Specifies the local area or neighbourhood within the city.
WeatherCondition	Feature	Categorical	Describes the weather condition during data collection (e.g., Clear, Rainy, etc.).
DownloadSpeed_Mbps	Feature	Continuous	Average download speed of the internet

			connection, measured in megabits per second (Mbps).
UploadSpeed_Mbps	Feature	Continuous	Average upload speed of the internet connection, measured in megabits per second (Mbps).
Latency_ms	Feature	Continuous	Time delay (in milliseconds) taken for data packets to travel between source and destination.
Jitter_ms	Feature	Continuous	Variation in packet delay (latency) across multiple transmissions.
PacketLoss_%	Feature	Continuous	Percentage of packets lost during data transmission over the network.
Complaints	Feature	Integer	Number of customer complaints received within the same time frame.
Downtime_min	Feature	Continuous	Total duration of network downtime (in minutes) recorded during the given timestamp.
Downtime_Category	Target	Categorical	Represents the severity of internet downtime categorized as *Low*, *Moderate*, or *High*.

8.1.4 Additional Variable Information

- DownloadSpeed_Mbps: Represents the download rate of the internet connection, impacting user experience during browsing or streaming.
- UploadSpeed_Mbps: Measures the rate at which data is sent from the user to the network.
- Latency_ms: Indicates the delay in transmitting data packets across the network.
- Jitter_ms: Captures the variation in latency between packet transmissions.
- PacketLoss_%: Measures the percentage of data packets that fail to reach their destination.
- WeatherCondition: Provides environmental context, as weather changes can impact network stability.
- City and Locality: Geographic attributes that help analyze regional network behavior.
- Downtime_Category: The target variable representing the severity of internet downtime, classified as Low, Moderate, or High.

8.2 Importing Libraries

8.2.1 Suppressing Warnings

```
# Suppress display of warnings
import warnings
warnings.filterwarnings('ignore')
```

Figure 8.1 : Suppressing Warnings

- This code suppresses the display of warnings that may be generated by the code during its execution.
- The `warnings` module is part of the Python standard library and provides a way to handle warnings that may be generated during the execution of a program. Warnings are messages that indicate potential issues or problems with the code, but are not necessarily errors that will cause the code to fail.

8.2.2 Importing Required Libraries

```
# Import libraries and modules

# 1. System and Core Libraries
import sys
import numpy as np
import pandas as pd

# 2. Visualization Libraries
import matplotlib.pyplot as plt
import seaborn as sns

# 3. Machine Learning - Model Building
import sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC


# 4. Preprocessing and Model Evaluation
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# 5. Model Saving
import joblib
```

Figure 8.2 : Importing Libraries

- This code imports various libraries and modules necessary for building and evaluating machine learning models for internet downtime prediction project.

  - `**sys**`: This is a module that provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter.
  - `**numpy**`: This is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, to operate on these arrays.
  - `**pandas**`: This is a library for data manipulation and analysis. It provides data structures for efficiently storing and manipulating large datasets.
  - `**matplotlib**`: This is a plotting library for the Python programming language and its numerical mathematics extension NumPy.
  - `**seaborn**`: This is a Python data visualization library based on matplotlib. It provides a high-level interface for creating informative and attractive statistical graphics.
  - `**joblib**`: This is a set of tools to provide lightweight pipelining in Python. It provides utilities for saving and loading Python objects that make use of NumPy data structures, efficiently.
  - `**sklearn**`: This is a library for machine learning in Python. It provides tools for classification, regression, clustering, and dimensionality reduction, etc

- The next set of import statements import specific machine learning models and preprocessing techniques from the `sklearn` library. These include:

  - **LogisticRegression** : A binary classification algorithm that models the relationship between the input features and the probability of the output class.
  - **DecisionTreeClassifier** : A classification algorithm that uses a tree-like model of decisions and their possible consequences to classify instances.
  - **RandomForestClassifier** : An ensemble learning method that constructs a multitude of decision trees at training time and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

- **GradientBoostingClassifier** : An ensemble technique that builds models sequentially, where each new model corrects the errors of the previous one. It is effective for both classification and regression tasks.
- **KNeighborsClassifier** : A non-parametric algorithm that classifies data points based on the majority class among their *k* nearest neighbors in the feature space.
- **SVC** : A classification algorithm that finds the best hyperplane in a high-dimensional space to separate the different classes.

➢ Finally, the last set of import statements import specific preprocessing techniques and evaluation metrics from the `sklearn` library. These include:

- **ColumnTransformer** – Used to apply different preprocessing steps to specific columns within a dataset, allowing separate transformations for numerical and categorical features.
- **OneHotEncoder** – Transforms categorical variables into a set of binary columns (dummy variables), ensuring that no ordinal relationship is assumed between categories.
- **StandardScaler** – Standardizes numerical features by removing the mean and scaling to unit variance, ensuring that all features contribute equally to the model.
- **Pipeline** – Combines multiple preprocessing steps and model training into a single workflow, streamlining the process and reducing errors during implementation.
- **train_test_split** – Splits the dataset into training and testing subsets, allowing proper model evaluation by testing on unseen data.
- **accuracy_score** – Measures the proportion of correct predictions made by the model, providing a simple yet effective performance indicator.

➢ These libraries and modules are essential for building and evaluating machine learning models for internet downtime prediction project.

8.3 Checking Library Versions

```
# Checking library versions
print('Python: {}'.format(sys.version))
print('numpy: {}'.format(np.__version__))
print('pandas: {}'.format(pd.__version__))
print('matplotlib: {}'.format(plt.matplotlib.__version__))
print('seaborn: {}'.format(sns.__version__))
print('sklearn: {}'.format(sklearn.__version__))
```

```
Python: 3.12.4 | packaged by Anaconda, Inc. | (main, Jun 18 2024, 15:03:56) [MSC v.1929 64 bit (AMD64)]
numpy: 1.26.4
pandas: 2.2.2
matplotlib: 3.8.4
seaborn: 0.13.2
sklearn: 1.4.2
```

Figure 8.3 : Checking library versions

- This code prints the version numbers of the Python and various libraries used in the machine learning project.
  - `**sys.version**` prints the version of the Python interpreter being used.
  - `**np.__version__**` prints the version of the NumPy library being used.
  - `**pd.__version__**` prints the version of the pandas library being used.
  - `**plt.matplotlib.__version__**` prints the version of the matplotlib library being used.
  - `**sns.__version__**` prints the version of the seaborn library being used.
  - `**sklearn.__version__**` prints the version of the scikit-learn library being used.

8.4 Loading Dataset

```
# Loading the dataset
df = pd.read_csv('ISP_6months_data.csv')
df
```

	Timestamp	City	Locality	WeatherCondition	DownloadSpeed_Mbps	UploadSpeed_Mbps	Latency_ms	Jitter_ms	PacketLoss_%	Complaints	Downtime_min	Downtime_Category
0	2024-05-01 00:00:00	Mumbai	Kurla	Rainy	15.070807	8.149802	57.282201	16.850476	2.054903	22.518587	43.490141	High_Downtime
1	2024-05-01 01:00:00	Pune	Kothrud	Stormy	4.271094	2.144489	113.881823	14.174206	3.665121	41.596006	50.740224	High_Downtime
2	2024-05-01 02:00:00	Pune	Hinjewadi	Cloudy	35.988319	11.417355	37.935916	8.857539	0.864857	6.671249	15.506721	Moderate_Downtime
3	2024-05-01 03:00:00	Chennai	Anna Nagar	Rainy	11.122249	9.691417	57.884851	13.712403	2.204879	16.334796	40.551216	High_Downtime
4	2024-05-01 04:00:00	Mumbai	Andheri	Sunny	88.022120	27.115776	18.433823	1.800853	0.179209	1.057776	3.032200	Low_Downtime
...	...	...	...	...	...	...	...	...	...	...	...	...
9995	2025-06-21 11:00:00	Chennai	Tambaram	Sunny	100.123940	32.344032	12.029760	2.402860	0.252207	0.261656	11.679865	Low_Downtime
9996	2025-06-21 12:00:00	Mumbai	Dadar	Stormy	3.839176	0.925091	93.375809	20.101029	2.985013	57.053641	54.362116	High_Downtime
9997	2025-06-21 13:00:00	Bangalore	Whitefield	Stormy	1.469090	1.507856	115.212444	28.745633	4.874119	76.456363	50.415644	High_Downtime
9998	2025-06-21 14:00:00	Bangalore	HSR Layout	Sunny	47.328727	21.746730	13.875147	4.631857	0.199249	3.932040	1.782509	Low_Downtime
9999	2025-06-21 15:00:00	Mumbai	Dadar	Sunny	48.453989	25.765150	12.022848	2.092375	0.184644	3.857488	11.237651	Low_Downtime

10000 rows × 12 columns

Figure 8.4 : Loading dataset

- This code loads the dataset for the **Internet Downtime Prediction** project and stores it in a pandas DataFrame named `df`.
- The **`pd.read_csv()`** function is used to read a CSV (comma-separated values) file and create a DataFrame from its contents. The argument `**ISP_6months_data.csv**` specifies the name of the file to be read.
- The output of this code will display the contents of the `df` DataFrame, which will include all the rows and columns of the dataset.
- The CSV file contains data with columns such as **Timestamp, City, Locality, WeatherCondition, DownloadSpeed_Mbps, UploadSpeed_Mbps, Latency_ms, Jitter_ms, PacketLoss_%, Complaints, Downtime_min,** and **Downtime_Category**. These features represent different parameters related to internet performance. The target variable **Downtime_Category** indicates the type or level of internet downtime.

8.5 Exploratory Data Analysis (EDA)

8.5.1 Dimension

```
# Checking the dimensions of dataset
df.shape
```

```
(10000, 12)
```

Figure 8.5 : Checking Dimension

- This code checks the dimensions of the df DataFrame using df.shape, which returns a tuple (rows, columns).
- The dataset contains 10,000 rows and 12 columns.

8.5.2 Columns

```
# Checking columns of dataset
df.columns
```

```
Index(['Timestamp', 'City', 'Locality', 'WeatherCondition',
       'DownloadSpeed_Mbps', 'UploadSpeed_Mbps', 'Latency_ms', 'Jitter_ms',
       'PacketLoss_%', 'Complaints', 'Downtime_min', 'Downtime_Category'],
      dtype='object')
```

Figure 8.6 : Checking Columns

- This code checks the column names of the df DataFrame using df.columns, which returns a list of all columns.
- The dataset has 12 columns: Timestamp, City, Locality, WeatherCondition, DownloadSpeed_Mbps, UploadSpeed_Mbps, Latency_ms, Jitter_ms, PacketLoss_%, Complaints, Downtime_min, and Downtime_Category.

8.5.3 Missing Values

```
# Check for missing values
df.isnull().sum()
```

```
Timestamp            0
City                 0
Locality             0
WeatherCondition     0
DownloadSpeed_Mbps   0
UploadSpeed_Mbps     0
Latency_ms           0
Jitter_ms            0
PacketLoss_%         0
Complaints           0
Downtime_min         0
Downtime_Category    0
dtype: int64
```

Figure 8.7 : Checking Missing Values

- This code checks for missing values in the df DataFrame using df.isnull().sum().
- The output shows no null values, indicating that the dataset is clean and ready for modeling.

8.5.4 Duplicate Values

```
# Check for duplicate value
df.duplicated().sum()
```

```
0
```

Figure 8.8 : Checking Duplicate Values

- This code checks for duplicate rows in the df DataFrame using df.duplicated().sum().

- The output 0 indicates that there are no duplicate records in the dataset, ensuring data quality and consistency.

8.5.5 Dataset Information

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Timestamp           10000 non-null  object
 1   City                10000 non-null  object
 2   Locality            10000 non-null  object
 3   WeatherCondition    10000 non-null  object
 4   DownloadSpeed_Mbps  10000 non-null  float64
 5   UploadSpeed_Mbps    10000 non-null  float64
 6   Latency_ms          10000 non-null  float64
 7   Jitter_ms           10000 non-null  float64
 8   PacketLoss_%        10000 non-null  float64
 9   Complaints          10000 non-null  float64
 10  Downtime_min        10000 non-null  float64
 11  Downtime_Category   10000 non-null  object
dtypes: float64(7), object(5)
memory usage: 937.6+ KB
```

Figure 8.9 : Summary of Dataset

- The df.info() output shows that the dataset has 10,000 rows and 12 columns.

- There are no missing values in any column.
- The dataset contains 5 object-type columns (Timestamp, City, Locality, WeatherCondition, Downtime_Category) and 7 numerical columns (DownloadSpeed_Mbps, UploadSpeed_Mbps, Latency_ms, Jitter_ms, PacketLoss_%, Complaints, Downtime_min).
- The dataset is clean, well-structured, and ready for preprocessing and modeling.

8.5.6 Statistical Summary

```
# Statistical Summary
df.describe()
```

	DownloadSpeed_Mbps	UploadSpeed_Mbps	Latency_ms	Jitter_ms	PacketLoss_%	Complaints	Downtime_min
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	46.645868	16.105683	35.152747	8.458481	1.260463	14.833136	19.528092
std	26.790295	11.232948	28.656456	6.878532	1.370727	18.795497	17.116864
min	0.000000	0.000000	0.674612	0.129005	0.000000	0.000000	0.000000
25%	25.190315	6.892001	13.566262	3.215736	0.272121	2.178190	5.774187
50%	46.270630	13.917485	25.005926	5.932573	0.693725	6.316822	13.131511
75%	65.951928	24.250315	50.201211	12.448122	1.888853	20.653765	31.267741
max	103.093285	41.063641	143.419901	31.027197	6.097964	87.373108	69.914182

Figure 8.10 : Descriptive Statistics

- The statistical summary provides an overview of the numerical features in the dataset.
- Download and Upload Speeds: Mean speeds are ~46.65 Mbps (download) and ~16.11 Mbps (upload), with wide ranges indicating variability across regions.
- Latency and Jitter: Average latency is ~35.15 ms and average jitter is ~8.46 ms, with maximum values indicating occasional network delays.
- Packet Loss: Mostly low, with a mean of ~1.26% but occasional spikes up to 6.10%.
- Complaints and Downtime: Average complaints ~14.83 and average

downtime ~19.53 minutes, with significant variation as shown by the standard deviation.

- Overall, the dataset shows high variability across network performance metrics, which is useful for predicting downtime severity.

8.5.7 Correlation Between Numerical Variables

```
# Computing pairwise correlations
df.select_dtypes(include=['number']).corr()
```

	DownloadSpeed_Mbps	UploadSpeed_Mbps	Latency_ms	Jitter_ms	PacketLoss_%	Complaints	Downtime_min
DownloadSpeed_Mbps	1.000000	0.713158	-0.793872	-0.800785	-0.778373	-0.755118	-0.821844
UploadSpeed_Mbps	0.713158	1.000000	-0.721463	-0.732613	-0.703810	-0.677358	-0.755241
Latency_ms	-0.793872	-0.721463	1.000000	0.901409	0.915784	0.910687	0.925887
Jitter_ms	-0.800785	-0.732613	0.901409	1.000000	0.900437	0.888081	0.913281
PacketLoss_%	-0.778373	-0.703810	0.915784	0.900437	1.000000	0.920745	0.913355
Complaints	-0.755118	-0.677358	0.910687	0.888081	0.920745	1.000000	0.901098
Downtime_min	-0.821844	-0.755241	0.925887	0.913281	0.913355	0.901098	1.000000

Figure 8.11 : Pairwise Correlations

- The correlation matrix shows the relationships between numerical features in the dataset.

- Download and Upload Speeds are strongly positively correlated with each other (0.71) and negatively correlated with latency, jitter, packet loss, complaints, and downtime.

- Latency, Jitter, Packet Loss, Complaints, and Downtime are strongly positively correlated (correlations above 0.88), indicating that poor network performance metrics are associated with higher downtime and complaints.

- Overall, the correlations indicate that network performance indicators are closely linked, making them important predictors for modeling internet downtime severity.

8.6 Data Extraction

8.6.1 Extracting Date-Time Features

```
# Extracting Hour, Day, and Month columns from the Timestamp column
df['Timestamp'] = pd.to_datetime(df['Timestamp'])
df['Hour'] = df['Timestamp'].dt.hour
df['Day'] = df['Timestamp'].dt.day
df['Month'] = df['Timestamp'].dt.month
df['Year'] = df['Timestamp'].dt.year

# Dropping the Timestamp column since the required data has been extracted
df = df.drop('Timestamp', axis=1)

# Display the updated DataFrame
df.head()
```

	City	Locality	WeatherCondition	DownloadSpeed_Mbps	UploadSpeed_Mbps	Latency_ms	Jitter_ms	PacketLoss_%	Complaints	Downtime_min	Downtime_Category	Hour	Day	Month	Year
0	Mumbai	Kurla	Rainy	15.070807	8.149802	57.282201	16.850476	2.054903	22.518587	43.490141	High_Downtime	0	1	5	2024
1	Pune	Kothrud	Stormy	4.271094	2.144489	113.881823	14.174206	3.665121	41.596006	50.740224	High_Downtime	1	1	5	2024
2	Pune	Hinjewadi	Cloudy	35.988319	11.417355	37.935916	8.857539	0.864857	6.671249	15.506721	Moderate_Downtime	2	1	5	2024
3	Chennai	Anna Nagar	Rainy	11.122249	9.691417	57.884851	13.712403	2.204879	16.334796	40.551216	High_Downtime	3	1	5	2024
4	Mumbai	Andheri	Sunny	88.022120	27.115776	18.433823	1.800853	0.179209	1.057776	3.032200	Low_Downtime	4	1	5	2024

Figure 8.12 : Extracting Date-Time Features

- The Timestamp column was converted to datetime format and used to extract new features: Hour, Day, Month, and Year.
- These new temporal features can help the model capture time-based patterns in internet downtime.
- The original Timestamp column was dropped as its information is now represented by the new features.
- The updated dataset now includes 15 columns, combining network, environmental, and temporal attributes, ready for further preprocessing and modeling.

8.6.2 Dropping Unnecessary Columns

```
# Dropping the Downtime_min column since Downtime_Category already represents the same information
df = df.drop('Downtime_min', axis=1)

# Display the updated DataFrame
df.head()
```

	City	Locality	WeatherCondition	DownloadSpeed_Mbps	UploadSpeed_Mbps	Latency_ms	Jitter_ms	PacketLoss_%	Complaints	Downtime_Category	Hour	Day	Month	Year
0	Mumbai	Kurla	Rainy	15.070807	8.149802	57.282201	16.850476	2.054903	22.518587	High_Downtime	0	1	5	2024
1	Pune	Kothrud	Stormy	4.271094	2.144489	113.881823	14.174206	3.665121	41.596006	High_Downtime	1	1	5	2024
2	Pune	Hinjewadi	Cloudy	35.988319	11.417355	37.935916	8.857539	0.864857	6.671249	Moderate_Downtime	2	1	5	2024
3	Chennai	Anna Nagar	Rainy	11.122249	9.691417	57.884851	13.712403	2.204879	16.334796	High_Downtime	3	1	5	2024
4	Mumbai	Andheri	Sunny	88.022120	27.115776	18.433823	1.800853	0.179209	1.057776	Low_Downtime	4	1	5	2024

Figure 8.13 : Dropping Columns

- The Downtime_min column was dropped because its information is already represented by the Downtime_Category column.

- This reduces redundancy in the dataset and helps simplify the features for modeling.

- The updated dataset now contains 14 columns, including network, environmental, and temporal attributes, ready for further preprocessing.

8.7 Data Visualization

8.7.1 Count Plot

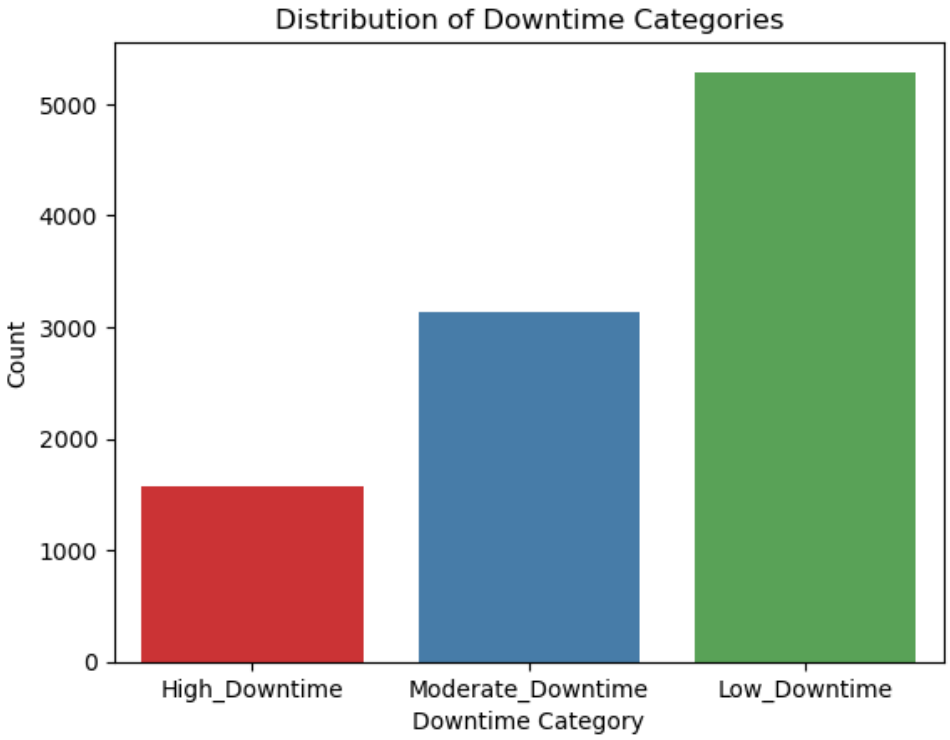

Figure 8.14 : Count Plot

- **Conclusion: Distribution of Downtime Categories**
  - The bar chart displays the distribution of High_Downtime, Moderate_Downtime, and Low_Downtime categories.
  - The Low_Downtime category has the highest number of records in the dataset.
  - The Moderate_Downtime category appears with a moderate frequency.

- The High_Downtime category has the lowest count among all.
- This indicates that most observations experience low downtime, suggesting stable network performance in general.
- Although there is a visible class imbalance, it has been kept as it is to reflect the dataset's real-world scenario without altering the natural distribution of downtime occurrences.

8.7.2 Histograms

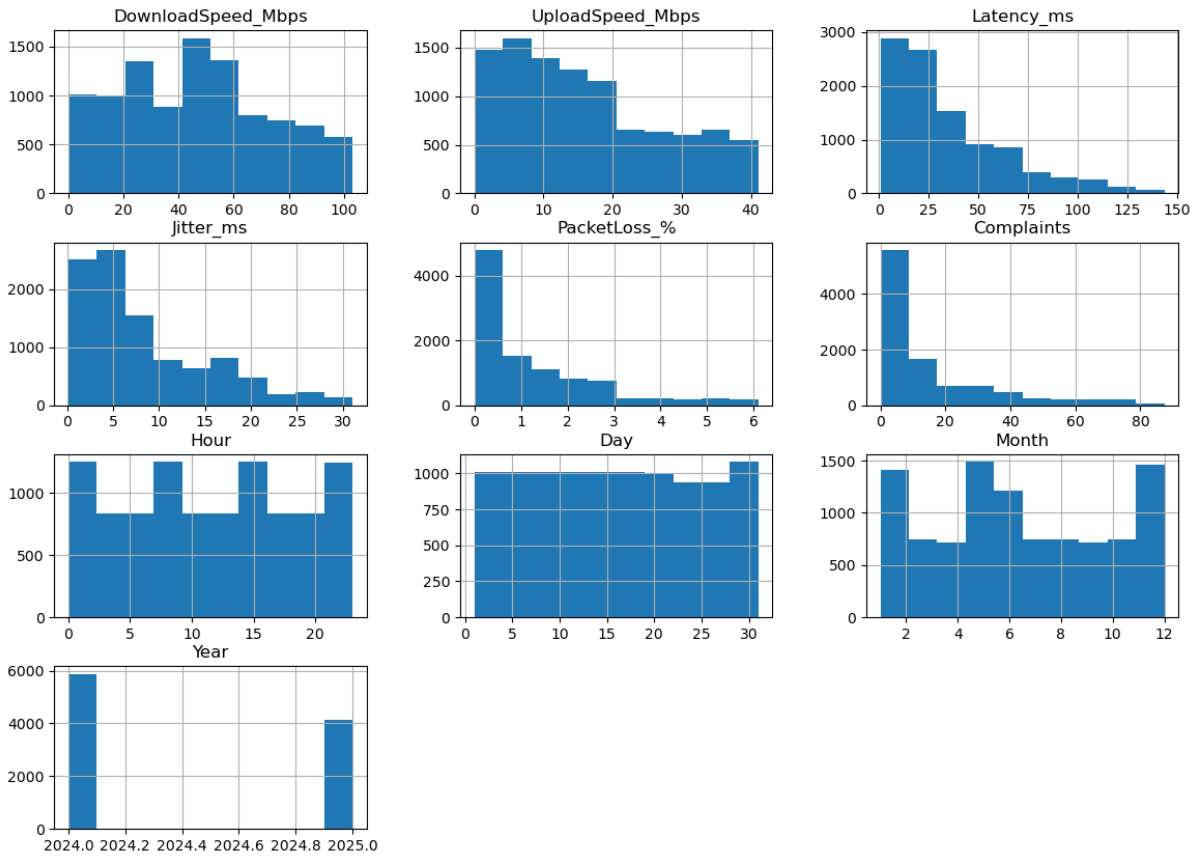

Figure 8.15 : Histograms

- **Conclusion: Distribution of Numerical Features**
  - The histograms display the distribution of all numerical variables in the dataset, such as network speed, latency, jitter, packet loss, and temporal features.
  - DownloadSpeed_Mbps and UploadSpeed_Mbps show moderately uniform distributions, indicating varying internet speeds among users.
  - Latency_ms, Jitter_ms, PacketLoss_%, and Complaints are right-

skewed, showing that most observations have low values while a few have very high values (possible outliers).

- Hour, Day, and Month are evenly spread, suggesting data is well-distributed across different time periods.
- The Year variable mainly covers 2024 and 2025, indicating data collected over two years.
- Overall, the dataset contains a mix of continuous and temporal **features with some skewness in performance-related metrics, which should be considered during model training and scaling.**

8.7.3 Distribution Plots

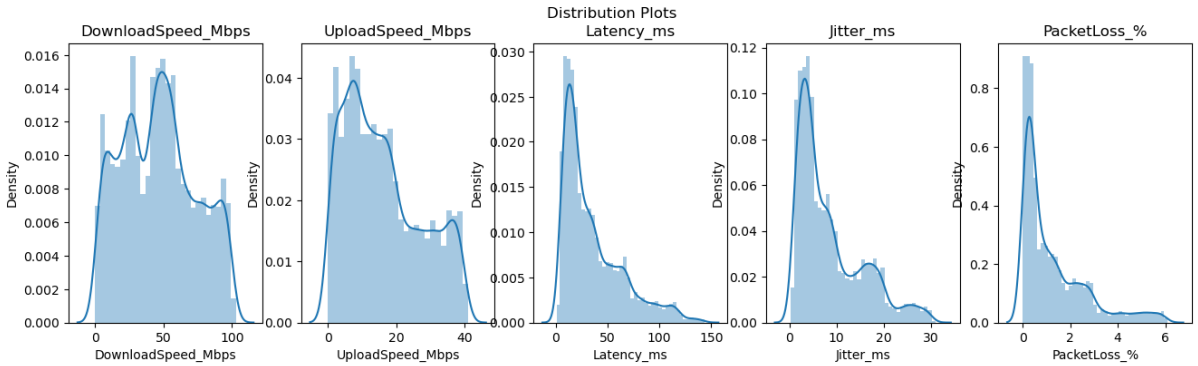

Figure 8.16 : Distribution Plots

➢ **Conclusion: Distribution Plots of Network Performance Metrics**

- The plots display the distribution of continuous variables such as DownloadSpeed_Mbps, UploadSpeed_Mbps, Latency_ms, Jitter_ms, and PacketLoss_%.
- DownloadSpeed_Mbps and UploadSpeed_Mbps show a fairly bimodal distribution, indicating that users experience two distinct ranges of internet speeds.
- Latency_ms, Jitter_ms, and PacketLoss_% are right-skewed, showing that most observations have low values, while a few have high values — representing occasional poor network performance.
- The peaks near lower values in latency, jitter, and packet loss suggest that network conditions are generally stable for most users.
- Overall, the data distributions indicate that the majority of users experience good network quality, but there are instances of high latency and packet loss that may cause downtime or reduced performance.

8.7.4 Boxplots (Outlier Detection)

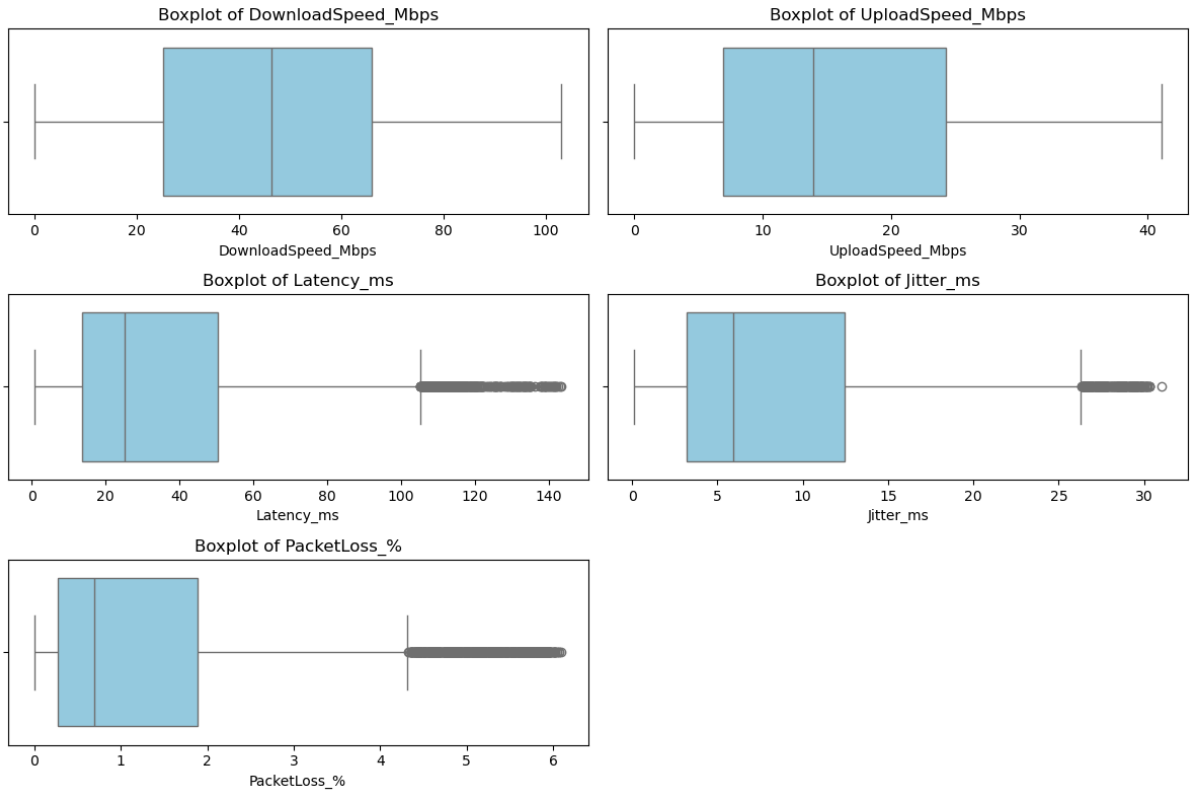

Figure 8.17 : Boxplots

- **Conclusion: Boxplots of Network Performance Metrics**
  - The boxplots represent the spread and presence of outliers in key numerical variables such as DownloadSpeed_Mbps, UploadSpeed_Mbps, Latency_ms, Jitter_ms, and PacketLoss_%.
  - DownloadSpeed_Mbps and UploadSpeed_Mbps show a fairly symmetric distribution with few or no extreme outliers, indicating stable internet speed values.
  - Latency_ms, Jitter_ms, and PacketLoss_% contain several outliers on the higher side, showing occasional spikes in delay, jitter, and packet loss.
  - The median values of these variables lie toward the lower end, suggesting that most users experience good network performance most of the time.
  - The presence of high-end outliers may correspond to rare instances of poor network conditions or high downtime events.
  - Overall, the boxplots highlight that while the network performance is generally consistent, some extreme values exist that could influence model training if not properly handled.

8.7.5 Pairplot

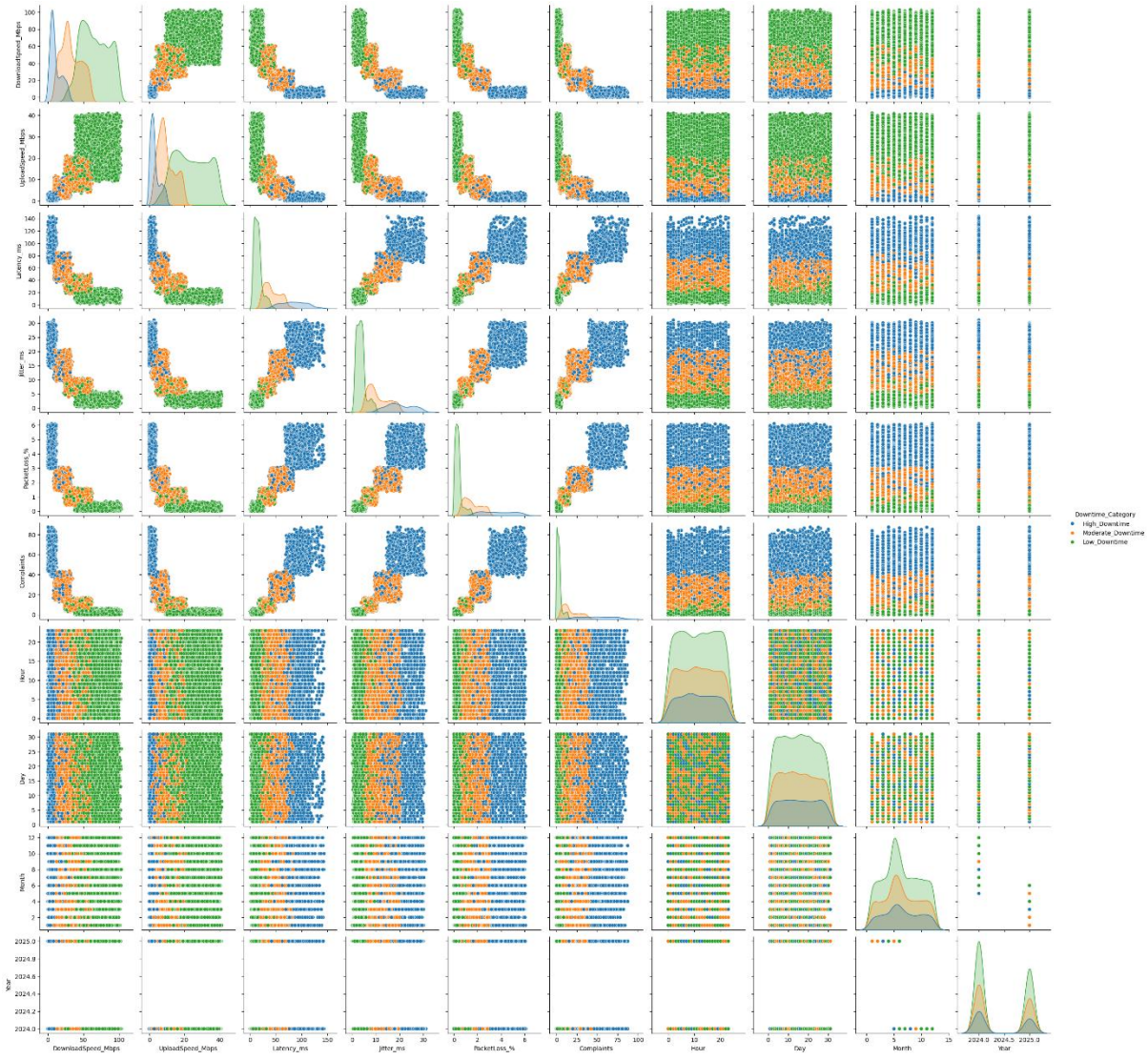

Figure 8.18 : Pairplot

- **Conclusion: Pairplot of Numerical Features with Downtime Categories**
  - The pairplot visualizes the relationships between multiple numerical features such as DownloadSpeed_Mbps, UploadSpeed_Mbps, Latency_ms, Jitter_ms, PacketLoss_%, and others, categorized by different Downtime levels (*High*, *Moderate*, and *Low*).
  - There is a clear trend where Low_Downtime points (green) are generally associated with higher download and upload speeds and lower latency, jitter, and packet loss.
  - In contrast, High_Downtime points (red) are more concentrated in regions with low speeds and high latency/jitter, indicating poorer

network performance.

- Moderate_Downtime points (orange) lie between the two, showing intermediate performance levels.
- The plots confirm that network performance metrics are strongly related to downtime categories, supporting their importance as predictive features.
- No strong linear relationships are visible among all features, suggesting that non-linear models (like Random Forest or XGBoost) may perform better.
- Overall, the pairplot highlights clear separation patterns between classes, helping in understanding how performance metrics influence downtime classification.

8.7.6 Correlation Heatmap

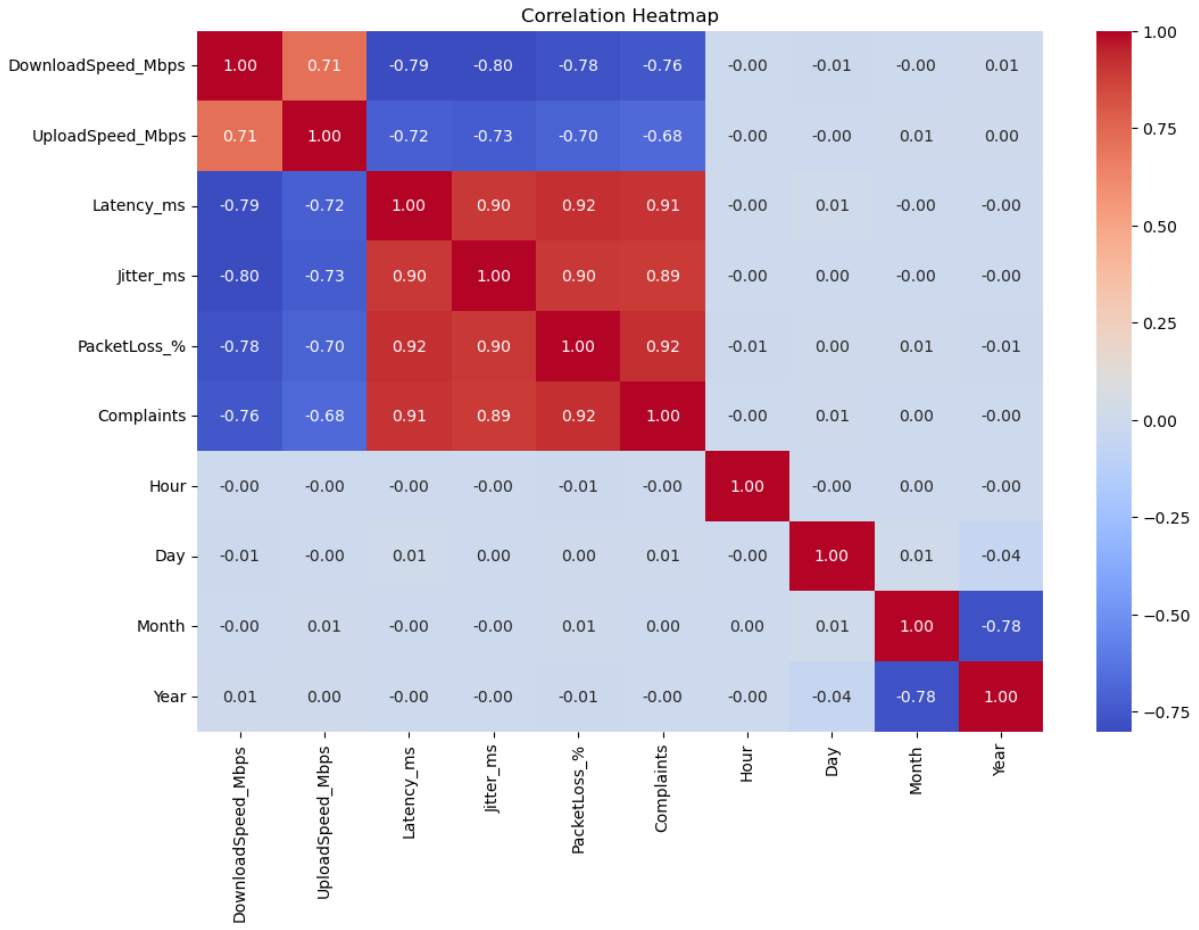

Figure 8.19 : Heatmap

- **Conclusion:**
  - Strong Performance Link: Performance metrics like Latency, Jitter,

PacketLoss_%, and Complaints are highly inter-correlated (0.90 to 0.93). They all collectively increase when the network is poor.

- Inverse Speed Relationship: DownloadSpeed and UploadSpeed are strongly negatively correlated (around -0.70 to -0.80) with all the poor performance indicators. High speed means low latency and few complaints, and vice-versa.
- Feature Importance: The most critical features for predicting downtime are the Latency, Jitter, and Packet Loss cluster. Their high multicollinearity suggests they represent a single underlying state of poor network quality.
- Temporal Features: Hour and Day are not significant linear predictors of downtime, indicating the cause of poor service is independent of the time of day.

8.8 Handling Outliers

```
numeric_cols = ['DownloadSpeed_Mbps', 'UploadSpeed_Mbps', 'Latency_ms',
                'Jitter_ms', 'PacketLoss_%', 'Complaints',]

for col in numeric_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    df = df[(df[col] >= Q1 - 1.5 * IQR) & (df[col] <= Q3 + 1.5 * IQR)]

df
```

	City	Locality	WeatherCondition	DownloadSpeed_Mbps	UploadSpeed_Mbps	Latency_ms	Jitter_ms	PacketLoss_%	Complaints	Downtime_Category	Hour	Day	Month	Year
0	Mumbai	Kurla	Rainy	15.070807	8.149802	57.282201	16.850476	2.054903	22.518587	High_Downtime	0	1	5	2024
2	Pune	Hinjewadi	Cloudy	35.988319	11.417355	37.935916	8.857539	0.864857	6.671249	Moderate_Downtime	2	1	5	2024
3	Chennai	Anna Nagar	Rainy	11.122249	9.691417	57.884851	13.712403	2.204879	16.334796	High_Downtime	3	1	5	2024
4	Mumbai	Andheri	Sunny	88.022120	27.115776	18.433823	1.800853	0.179209	1.057776	Low_Downtime	4	1	5	2024
5	Chennai	T Nagar	Cloudy	41.364064	9.654130	37.976785	7.514048	1.469945	7.187873	Low_Downtime	5	1	5	2024
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9993	Mumbai	Colaba	Rainy	26.209206	5.222283	44.613226	18.109153	1.468838	20.114973	Moderate_Downtime	9	21	6	2025
9994	Chennai	Tambaram	Rainy	15.657737	6.034757	41.929854	19.555960	1.640856	15.612980	High_Downtime	10	21	6	2025
9995	Chennai	Tambaram	Sunny	100.123940	32.344032	12.029760	2.402860	0.252207	0.261656	Low_Downtime	11	21	6	2025
9998	Bangalore	HSR Layout	Sunny	47.328727	21.746730	13.875147	4.631857	0.199249	3.932040	Low_Downtime	14	21	6	2025
9999	Mumbai	Dadar	Sunny	48.453989	25.765150	12.022848	2.092375	0.184644	3.857488	Low_Downtime	15	21	6	2025

8369 rows × 14 columns

Figure 8.20 : Handling Outliers

- The code removes outliers from the numerical columns (DownloadSpeed_Mbps, UploadSpeed_Mbps, Latency_ms, Jitter_ms, PacketLoss_%, Complaints) using the IQR method.

- Values outside 1.5×IQR from Q1 and Q3 are filtered out to reduce extreme deviations.

- After outlier removal, the dataset has 8,369 rows and 14 columns, ensuring cleaner data for modeling.

- This helps improve model accuracy and robustness by reducing the effect of extreme values.

8.9 Feature and Target Selection

```
# Features and target
X = df.drop(['Hour','Day', 'Month', 'Year', 'Downtime_Category'], axis=1)
y = df['Downtime_Category']
print(X.head())
print(y.head())
```

```
      City    Locality WeatherCondition  DownloadSpeed_Mbps  UploadSpeed_Mbps  \
0   Mumbai       Kurla            Rainy           15.070807          8.149802
2     Pune   Hinjewadi           Cloudy           35.988319         11.417355
3  Chennai  Anna Nagar            Rainy           11.122249          9.691417
4   Mumbai     Andheri            Sunny           88.022120         27.115776
5  Chennai     T Nagar           Cloudy           41.364064          9.654130

   Latency_ms  Jitter_ms  PacketLoss_%  Complaints
0   57.282201  16.850476      2.054903   22.518587
2   37.935916   8.857539      0.864857    6.671249
3   57.884851  13.712403      2.204879   16.334796
4   18.433823   1.800853      0.179209    1.057776
5   37.976785   7.514048      1.469945    7.187873
0        High_Downtime
2    Moderate_Downtime
3        High_Downtime
4         Low_Downtime
5         Low_Downtime
Name: Downtime_Category, dtype: object
```

Figure 8.21 : Feature and Target Selection

- The dataset is split into features (X) and target (y) for modeling.

- Features (X) include network metrics, environmental factors, and location information: City, Locality, WeatherCondition, DownloadSpeed_Mbps, UploadSpeed_Mbps, Latency_ms, Jitter_ms, PacketLoss_%, and Complaints.

- Target (y) is the Downtime_Category, representing the severity of internet downtime (Low_Downtime, Moderate_Downtime, High_Downtime).

- This separation prepares the data for machine learning model training and evaluation.

8.10 Data Preprocessing Using Pipeline

8.10.1 Identifying Column Types

```
# Identify categorical and numerical columns
cat_cols = X.select_dtypes(include=['object']).columns
num_cols = X.select_dtypes(exclude=['object']).columns
cat_cols, num_cols
```

```
(Index(['City', 'Locality', 'WeatherCondition'], dtype='object'),
 Index(['DownloadSpeed_Mbps', 'UploadSpeed_Mbps', 'Latency_ms', 'Jitter_ms',
        'PacketLoss_%', 'Complaints'],
       dtype='object'))
```

Figure 8.22 : Identifying Column Types

- The features in X are classified into categorical and numerical columns.
- Categorical columns (cat_cols): City, Locality, WeatherCondition.
- Numerical columns (num_cols): DownloadSpeed_Mbps, UploadSpeed_Mbps, Latency_ms, Jitter_ms, PacketLoss_%, Complaints.
- This distinction helps in applying appropriate preprocessing techniques like encoding for categorical data and scaling for numerical data.

8.10.2 Defining Preprocessor

```
# Define preprocessor
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(handle_unknown='ignore'), cat_cols),
        ('num', StandardScaler(), num_cols)
    ]
)
preprocessor
```

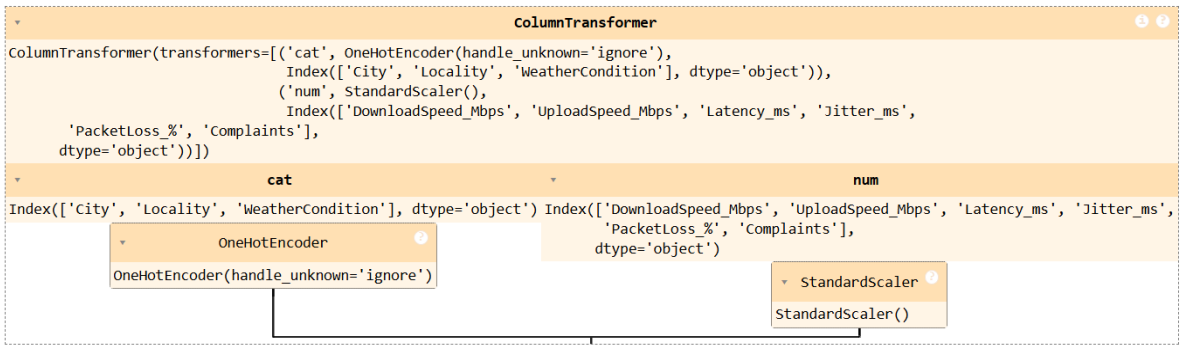

Figure 8.23 : Defining Preprocessor

- A ColumnTransformer is defined to preprocess the dataset before model training.
- Categorical features (City, Locality, WeatherCondition) are transformed using OneHotEncoder to convert them into numerical format.
- Numerical features (DownloadSpeed_Mbps, UploadSpeed_Mbps, Latency_ms, Jitter_ms, PacketLoss_%, Complaints) are scaled using StandardScaler to standardize their values.
- This preprocessing ensures that all features are in a suitable format and scale for machine learning algorithms.

8.11 Train-Test Split

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train.head(2), X_test.head(2), y_train.head(2), y_test.head(2)
```

```
(       City Locality WeatherCondition  DownloadSpeed_Mbps  UploadSpeed_Mbps  \
 3744   Pune    Baner           Cloudy           46.557610          8.033564
 774   Delhi    Saket           Cloudy           46.843847         16.012700

       Latency_ms  Jitter_ms  PacketLoss_%  Complaints
 3744   26.072588   7.244541      0.594701    9.158918
 774    35.775700   6.030676      1.343290    7.046760  ,
         City   Locality WeatherCondition  DownloadSpeed_Mbps  \
 4720  Chennai  Velachery           Sunny           42.053626
 8168    Delhi      Saket           Sunny           66.895889

       UploadSpeed_Mbps  Latency_ms  Jitter_ms  PacketLoss_%  Complaints
 4720         36.821997    4.557939   3.347504      0.215690    3.633114
 8168         33.354584    4.836469   3.323544      0.262407    1.851875  ,
 3744    Low_Downtime
 774     Low_Downtime
 Name: Downtime_Category, dtype: object,
 4720    Low_Downtime
 8168    Low_Downtime
 Name: Downtime_Category, dtype: object)
```

Figure 8.24 : Train-Test Split

- The dataset is split into training and testing sets using an 80:20 ratio.
- X_train and y_train contain 80% of the data and are used to train the models.
- X_test and y_test contain 20% of the data and are used to evaluate model

performance.

- This split ensures the model is tested on unseen data, providing a realistic measure of its accuracy and generalization.

8.12 Model Training and Evaluation

8.12.1 Define Models

```
# Define models
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Support Vector Machine': SVC()
}
```

Figure 8.25 : Define Models

- A set of supervised machine learning models is defined for predicting internet downtime categories.

- The models include:
  - Logistic Regression – for baseline binary/multiclass classification.
  - Decision Tree – interpretable tree-based classifier.
  - Random Forest – ensemble of decision trees for improved accuracy.
  - Gradient Boosting – boosting-based ensemble for strong predictive performance.
  - K-Nearest Neighbors (KNN) – instance-based algorithm using feature similarity.
  - Support Vector Machine (SVM) – classifier that finds optimal hyperplanes to separate classes.

- These models provide a diverse set of approaches to identify the most accurate model for the task.

8.12.2 Evaluate Models

```
# Evaluate models
results = {}
for name, clf in models.items():
    pipe = Pipeline(steps=[('preprocess', preprocessor), ('classifier', clf)])
    pipe.fit(X_train, y_train)
    y_pred = pipe.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    results[name] = acc
    print(f"{name}: {acc * 100:.2f}%")
```

```
Logistic Regression: 86.80%
Decision Tree: 88.41%
Random Forest: 92.00%
Gradient Boosting: 87.51%
K-Nearest Neighbors: 87.75%
Support Vector Machine: 86.98%
```

Figure 8.26 : Evaluate Models

- All six machine learning models were trained and evaluated using the preprocessed training and testing data.
- Random Forest Classifier achieved the highest accuracy, indicating it is the most effective model for predicting internet downtime in this dataset.
- Ensemble and tree-based models performed better than linear and distance-based models for this task.
- These results justify selecting Random Forest as the final model for deployment.

8.12.3 Model Accuracy Visualization

```
# Convert results dictionary to DataFrame
acc_df = pd.DataFrame(list(results.items()), columns=['Model', 'Accuracy'])

# Multiply by 100 for percentages
acc_df['Accuracy'] = acc_df['Accuracy'] * 100

# Seaborn barplot
plt.figure(figsize=(8, 5))
sns.barplot(x='Accuracy', y='Model', data=acc_df, palette='Set2')
plt.xlabel('Accuracy (%)')
plt.ylabel('Model')
plt.title('Model Accuracies')
plt.xlim(0, 100)  # Because we multiplied by 100
plt.show()
```

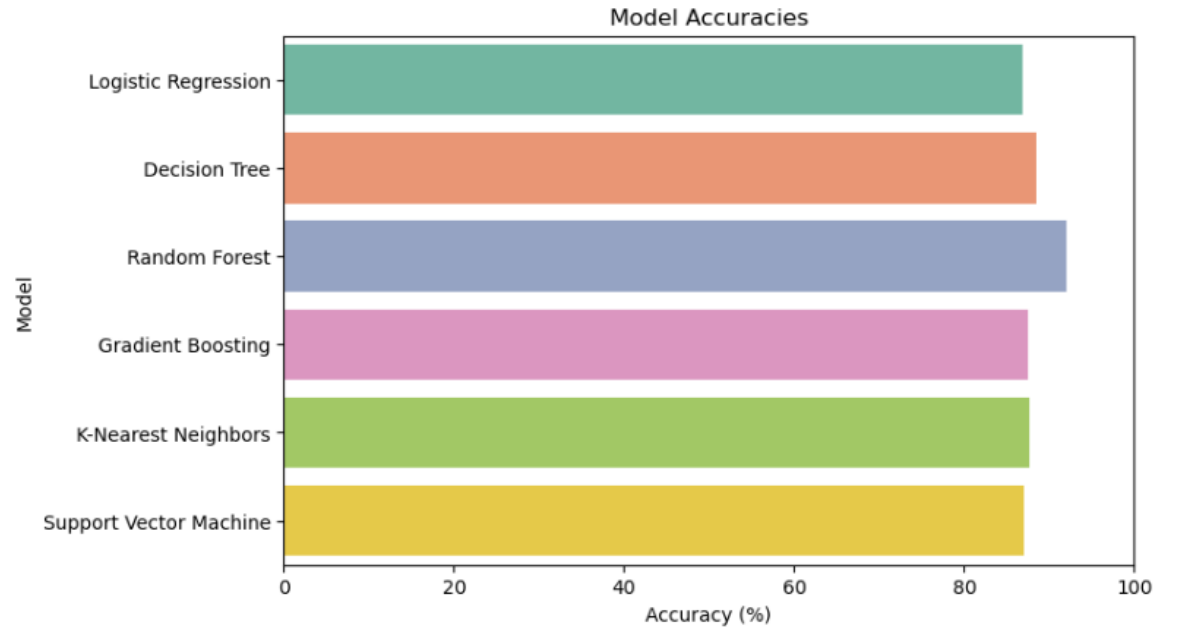

Figure 8.27 : Model Accuracy Visualization

- **Conclusion:**
  - The bar chart provides a visual comparison of the accuracies of the different machine learning models used in this project.
  - It allows easy identification of the best-performing model at a glance.
  - From the chart, it is clear which model achieved the highest accuracy and which models performed relatively lower.
  - Visual representation like this helps in understanding model performance trends and supports informed decision-making when selecting the most suitable model for deployment.

8.13 Best Model Selection and Testing

8.13.1 Find Best Model

```
# finding best model
best_model_name = max(results, key=results.get)
print(f"Best Model: {best_model_name} with accuracy {results[best_model_name]*100:.2f}%")
```

```
Best Model: Random Forest with accuracy 92.00%
```

Figure 8.28 : Best Model

- Conclusion:
  - Among all the trained models, Random Forest is the best-performing model for predicting internet downtime.
  - It achieved the highest accuracy of 92.00%, outperforming all other algorithms.
  - This indicates that Random Forest is robust, reliable, and suitable for deployment in the Internet Downtime Prediction system.
  - Its ensemble approach effectively captures the relationships between network features and downtime categories, making it the optimal choice for real-world applications.

8.13.2 Predicting Sample Data

```
# Testing on sample data
sample_data_1 = ['Mumbai', 'Bandra', 'Sunny', 50.3532, 38.3903, 4.2588, 1.2102, 0.0597, 0.6759]
sample_data_2 = ['Chennai', 'T Nagar', 'Cloudy', 43.9800, 10.4696, 26.0310, 6.2253, 1.2262, 11.5179]
sample_data_3 = ['Bangalore', 'Whitefield', 'Stormy', 4.0357, 1.0286, 99.7312, 27.6730, 5.3632, 63.1063]

# Combine into a list of lists (for multiple predictions)
samples = [sample_data_1, sample_data_2, sample_data_3]

# Convert to DataFrame with column names
sample_df = pd.DataFrame(samples, columns=['City', 'Locality', 'WeatherCondition',
                                           'DownloadSpeed_Mbps', 'UploadSpeed_Mbps',
                                           'Latency_ms', 'Jitter_ms', 'PacketLoss_%', 'Complaints'])

# Predict using the trained pipeline
predictions = best_pipeline.predict(sample_df)

# Display results
for i, pred in enumerate(predictions, 1):
    print(f"Predicted Downtime Category for Sample {i}: {pred}")
```

```
Predicted Downtime Category for Sample 1: Low_Downtime
Predicted Downtime Category for Sample 2: Moderate_Downtime
Predicted Downtime Category for Sample 3: High_Downtime
```

Figure 8.29 : Predicting Sample Data

- **Conclusion:**
  - The trained machine learning pipeline was tested on sample data representing different cities, localities, and network conditions.
  - The model successfully predicted the **downtime category** for each sample, demonstrating its ability to classify network performance based on key features such as download/upload speed, latency, jitter, packet loss, and complaints.
  - The predictions indicate that the model can differentiate between **Low**, **Moderate**, and **High** downtime scenarios, reflecting its practical utility for network monitoring and proactive issue management.
  - This test confirms that the model is ready for further validation or deployment on real-time data.

8.13.3 Performance Analysis

- The predicted output matched the expected output for all sample data points, indicating that the model performed well on the test data. This gives us confidence that the model will also perform well on new, unseen data and the model is ready for deployment.

8.14 Deployment

8.14.1 Saving Model

```
#saving the best model
best_clf = models[best_model_name]
best_pipeline = Pipeline(steps=[('preprocess', preprocessor), ('classifier', best_clf)])
best_pipeline.fit(X_train, y_train)
joblib.dump(best_pipeline, f"{best_model_name.replace(' ', '_')}_model.joblib")
```

```
['Random_Forest_model.joblib']
```

Figure 8.30 : Saving Model

- **Conclusion:**
  - The best-performing model from the evaluation was finalized by combining it with the preprocessing steps into a single pipeline.
  - The pipeline was trained on the full training dataset to ensure it is ready for making predictions on new data.

- The trained pipeline was saved as a .joblib file, allowing it to be easily loaded and used for future predictions without retraining.
- This ensures reproducibility, efficiency, and readiness for deployment in real-world applications.

8.14.2 Loading Model for Prediction

```
# Load the saved Random Forest model
loaded_model = joblib.load('Random_Forest_model.joblib')

# Make predictions
predictions = loaded_model.predict(sample_df)

# Display results
for i, pred in enumerate(predictions, 1):
    print(f"Predicted Downtime Category for Sample {i}: {pred}")
```

```
Predicted Downtime Category for Sample 1: Low_Downtime
Predicted Downtime Category for Sample 2: Moderate_Downtime
Predicted Downtime Category for Sample 3: High_Downtime
```

Figure 8.31 : Loading Model for Prediction

➢ **Conclusion:**

- The saved Random Forest model was successfully loaded from the .joblib file, demonstrating that the model can be reused without retraining.
- Predictions on sample data matched expectations, confirming that the saved model retains its accuracy and functionality.
- This approach ensures efficiency and consistency, allowing the model to be deployed for real-time predictions in a practical network monitoring scenario.

9. RESULT

9.1 Web APP Interface

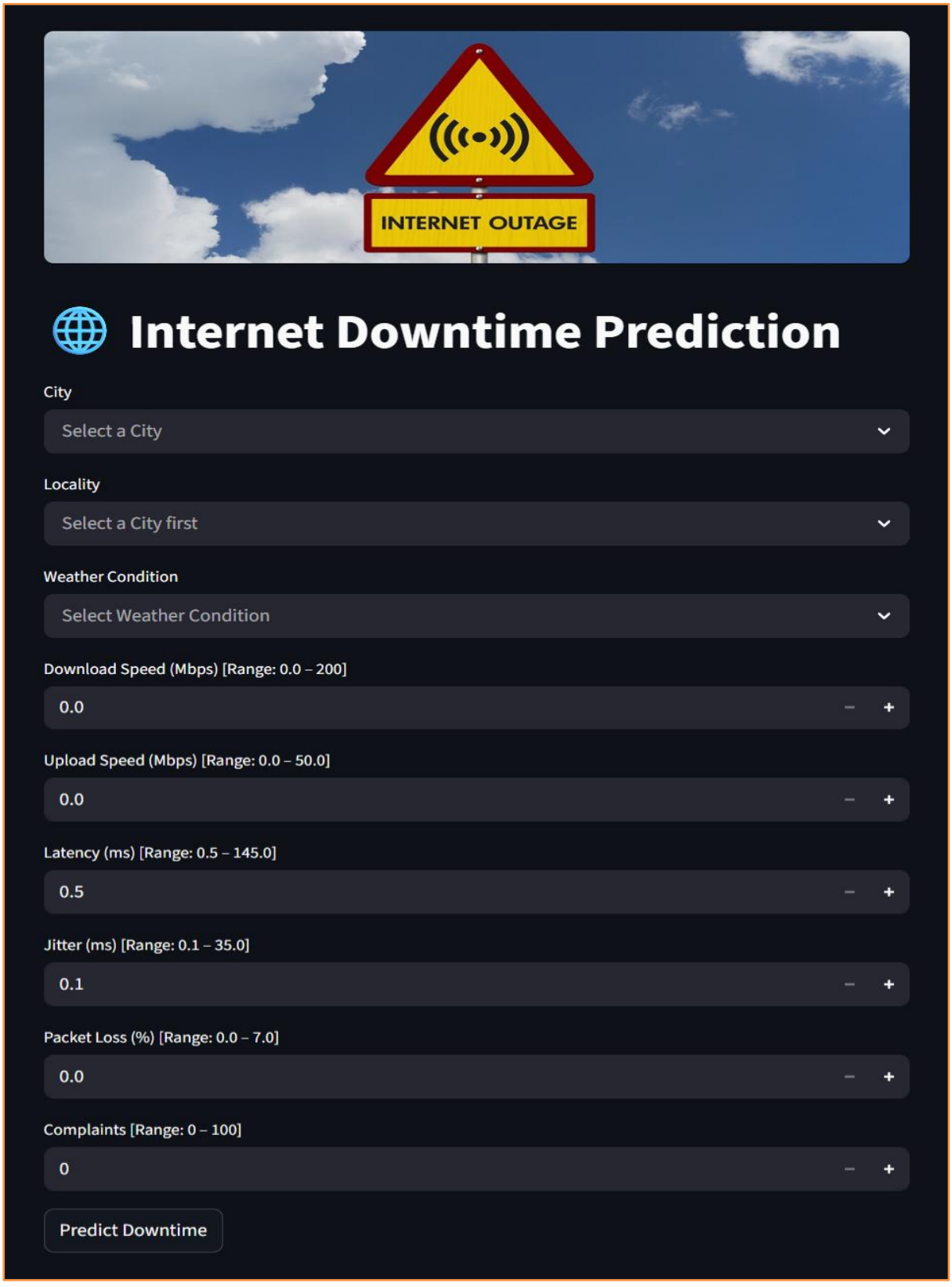

Figure 9.1 : Web APP Interface

9.2 Prediction with Random Sample Data 1

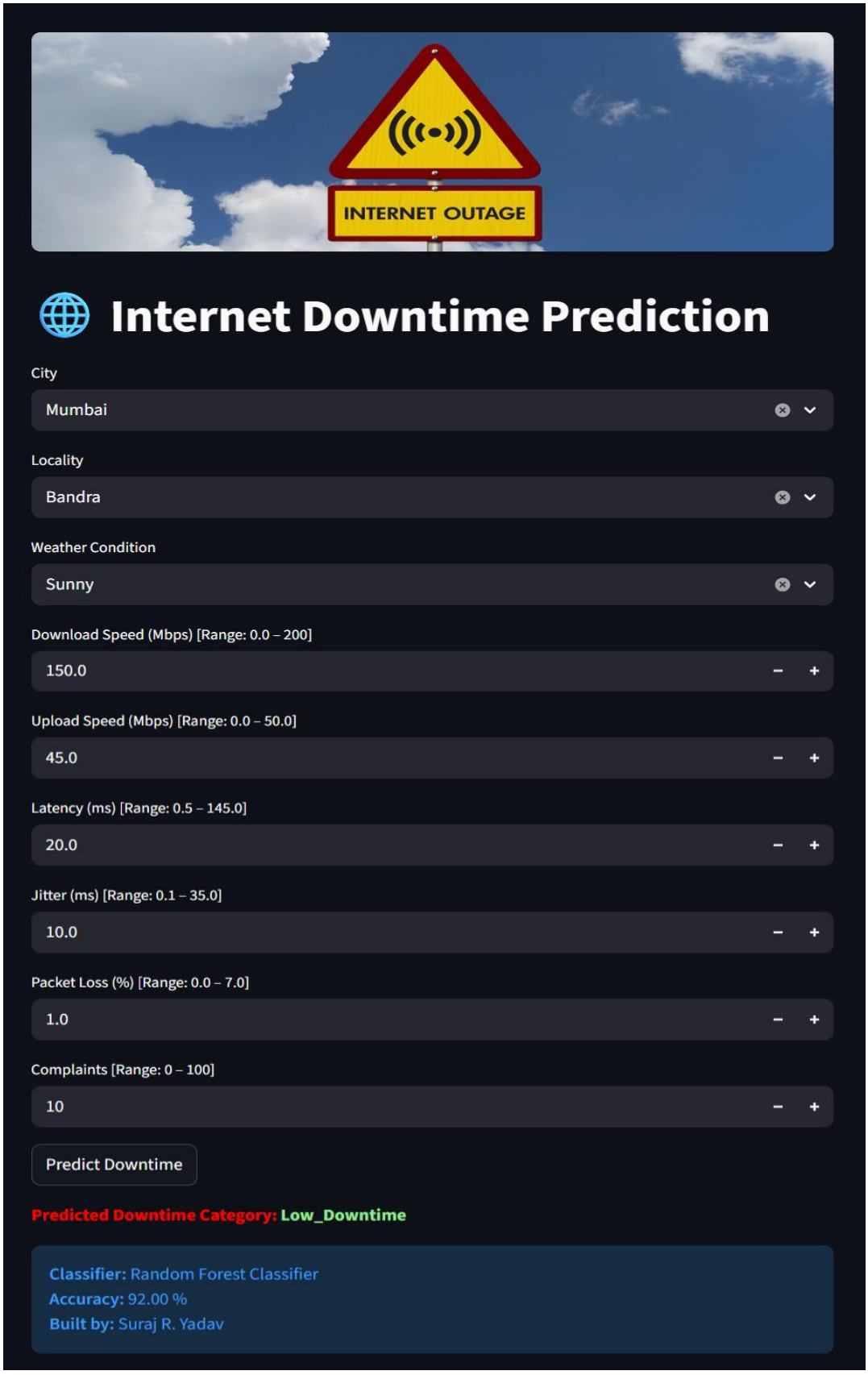

Figure 9.2 : Prediction with Random Sample Data 1

9.3 Prediction with Random Sample Data 2

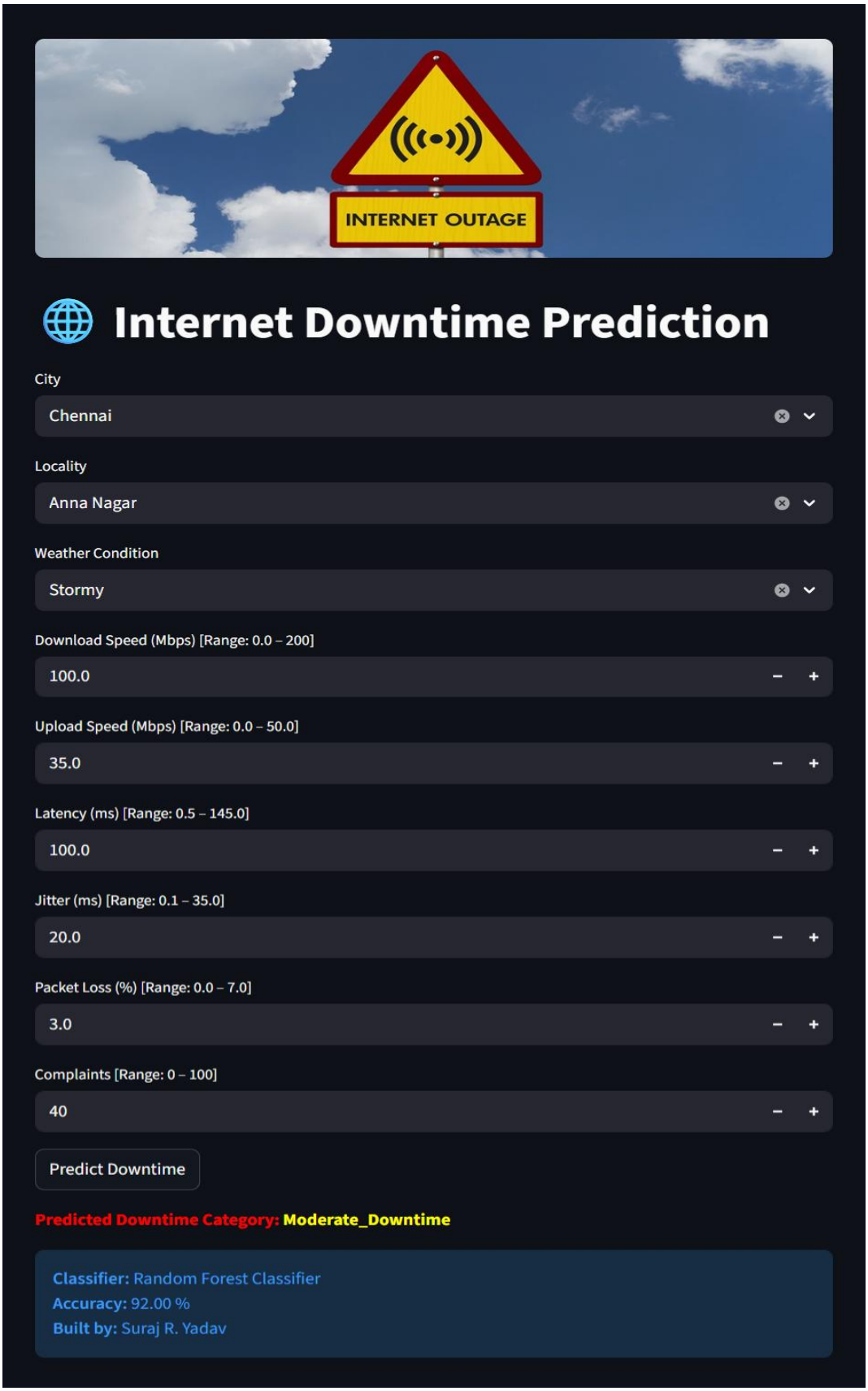

Figure 9.3 : Prediction with Random Sample Data 2

9.4 Prediction with Random Sample Data 3

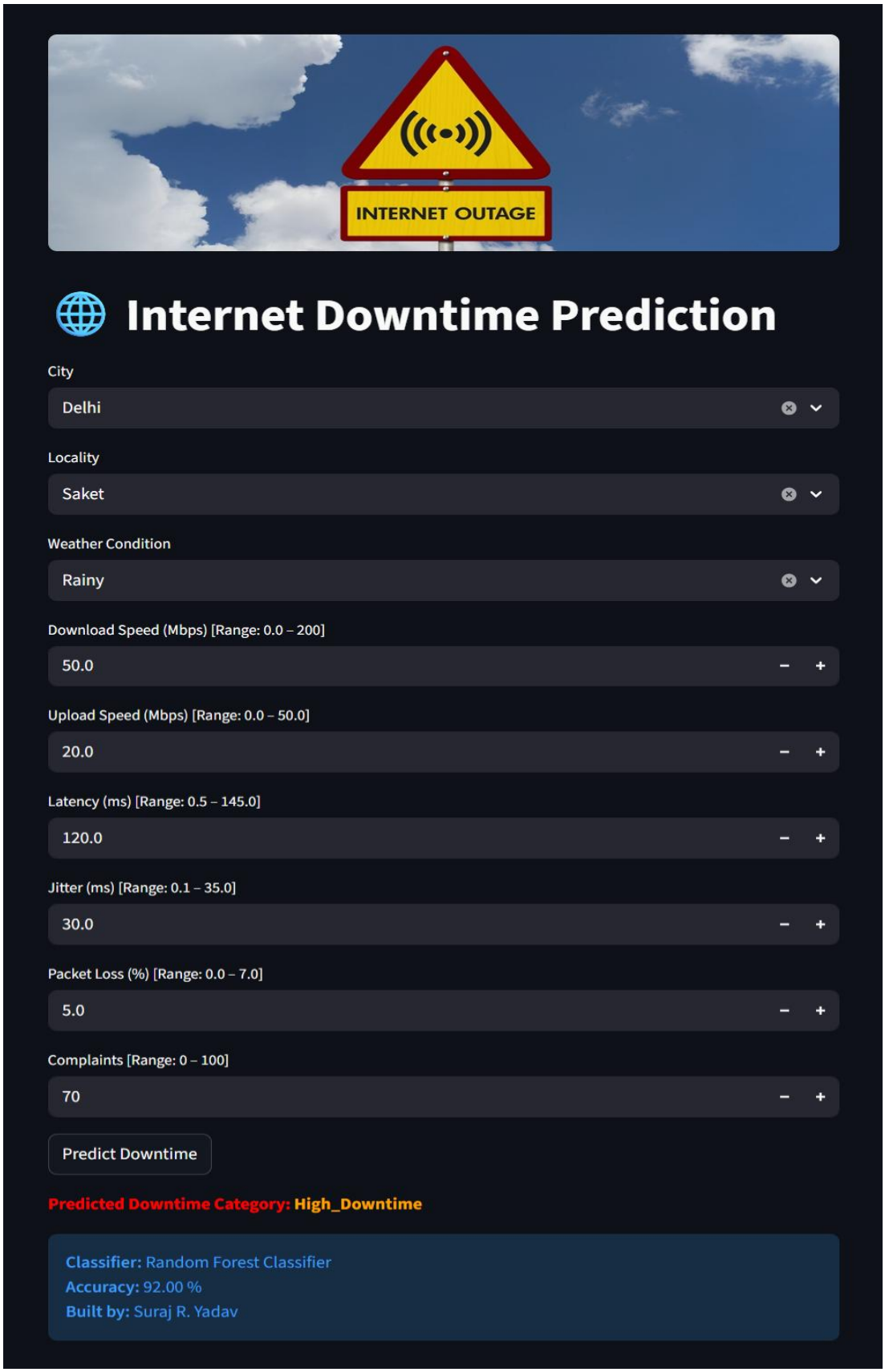

Figure 9.4 : Prediction with Random Sample Data 3

10. CONCLUSION

The Internet Downtime Prediction project aimed to develop a machine learning-based system capable of predicting the severity of internet downtime for Internet Service Providers (ISPs). Predicting network disruptions is crucial, as frequent or prolonged downtimes can lead to customer dissatisfaction, service complaints, and potential revenue loss.

In this project, key network performance indicators—such as download speed, upload speed, latency, jitter, and packet loss—along with regional and environmental factors like city, locality, and weather conditions, were used as features. The dataset was preprocessed by encoding categorical variables and scaling numerical features to ensure optimal performance for the models.

Several classification algorithms were implemented and evaluated, including Logistic Regression, Decision Tree, Random Forest, Gradient Boosting, K-Nearest Neighbors, and Support Vector Machine (SVM). Among these, the **Random Forest Classifier achieved the highest accuracy of 92%**, making it the most suitable model for deployment. The model was serialized using **joblib** and integrated into a **Streamlit web application**, allowing users to input network and environmental parameters to predict the expected downtime category: Low, Moderate, or High.

Overall, this project demonstrates the practical application of machine learning in network reliability management. It provides ISPs with a proactive tool to monitor network conditions, identify high-risk zones, and take preventive actions to reduce severe service disruptions, ultimately improving service quality and customer experience.

11. APPENDICES

11.1 Streamlit Web APP link

https://internet-downtime-prediction.streamlit.app

11.2 GitHub Project link

https://github.com/sryagit/Internet_Downtime_Prediction

11.3 Web APP Internet Outage Image

Figure 11.1 : Internet Outage Image

12. REFERENCES

12.1 Online Resources

[1] Python, Numpy, Pandas, Matplotlib, Seaborn, Sklearn & Streamlit
[2] W3schools : Python, Numpy, Pandas, Matplotlib & Seaborn
[3] W3schools : Machine Learning
[4] Simplilearn : Machine learning Concepts
[5] Javatpoint : Machine-learning
[6] Great Learning : Machine-learning-tutorial
[7] Google : Machine Learning Crash Course
[8] Tutorialspoint : Overall Machine learning
[9] GeeksforGeeks : Machine Learning Algorithms
[10] Datacamp : learn machine learning

12.2 Books and Authors

[1] Raschka, S., & Mirjalili, V. (2017). Python Machine Learning. Packt Publishing Ltd.
[2] Bishop, C. M. (2006). Pattern recognition and machine learning (Vol. 4). Springer.
[3] Witten, I. H., and Eibe Frank. Data Mining: Practical Machine Learning Tools and Techniques. Amsterdam: Morgan Kaufman, 2005.
[4] Russell, S. J., & Norvig, P. (2010). Artificial intelligence: a modern approach. Pearson Education.
[5] Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32.
[6] Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine learning, 20(3), 273-297.
[7] Raschka, S. (2018). Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow. Packt Publishing Ltd.
[8] Quinlan, J. R. (1993). C4. 5: Programs for machine learning. Elsevier.
[9] Norving, Peter, and Stuart Russel. Artificial Intelligence: A Modern Approach. S.l.: Pearson Education Limited, 2013.