PROJECT REPORT

BCA - Data Science | Sem 6 | Nov-Dec 2025

Name : SURAJ RAVISHANKAR YADAV

Reg. No. : EC2331201010110

Under the Guidance of

Prof. Muthuselvam

TOPIC

Internet Downtime Prediction Web APP
using
Python, Machine Learning, GitHub & Streamlit

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.

TOOLS & TECHNOLOGIES

- **Python**
  - NumPy
  - Pandas
  - Seaborn
  - Matplotlib
  - Scikit-learn

- **Anaconda Navigator**
- **Jupyter Notebook**
- **Excel**

- **Machine learning algorithms**
  - Logistic Regression
  - Decision Tree Classifier
  - Random Forest Classifier
  - Gradient Boosting Classifier
  - K-Nearest Neighbors Classifier
  - Support Vector Classifier (SVC)

- **Github**
- **Streamlit**
- **Notepad++**

IMPLEMENTATION

- ✓ Problem Statement
- ✓ Data Understanding
- ✓ Importing Libraries
- ✓ Loading the dataset
- ✓ Exploratory Data Analysis
- ✓ Data Visualization
- ✓ Data Cleaning
- ✓ Feature Selection
- ✓ Train-Test Split
- ✓ Feature Scaling
- ✓ Modeling Training
- ✓ Model Performance
- ✓ Performance Analysis
- ✓ Saving Model
- ✓ Loading Model
- ✓ User Interface Design
- ✓ Testing
- ✓ Deployment
- ✓ Result
- ✓ Conclusion

Importing 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
```

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

Suppressing Warnings

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
```

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

Exploratory Data Analysis (EDA)

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

```
(10000, 12)
```

Checking Dimensions

```
# 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')
```

Checking Columns

Duplicate Values

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

```
0
```

NULL 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
```

Summary of Dataset

```
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
```

Descriptive Statistics

```
# 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

```
# 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

Pairwise Correlations

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

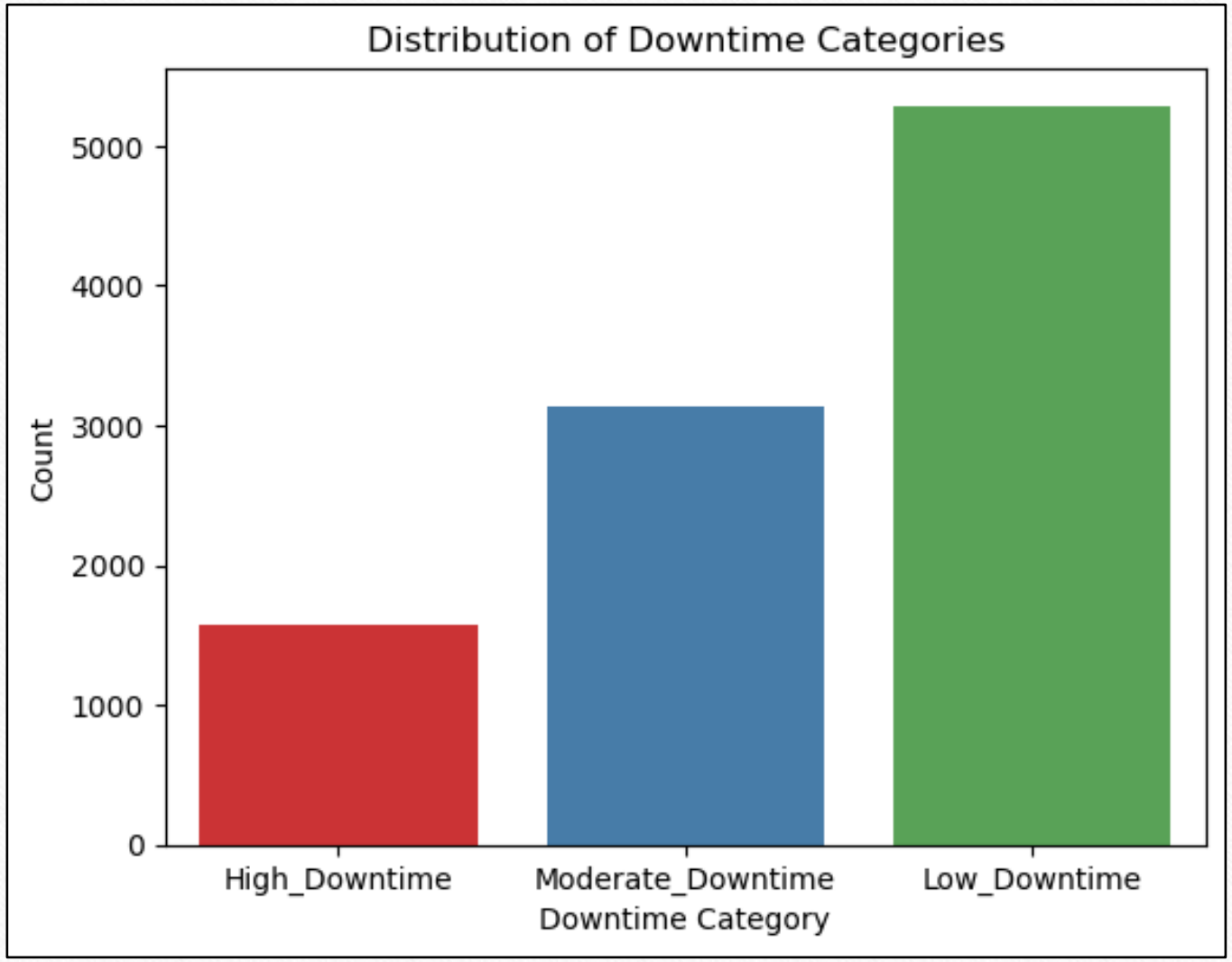
Dropping Unnecessary Columns

```
# 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

Count plot



Conclusion:

1) The bar chart displays the distribution of High_Downtime, Moderate_Downtime, and Low_Downtime categories.

2) The Low_Downtime category has the highest number of records in the dataset.

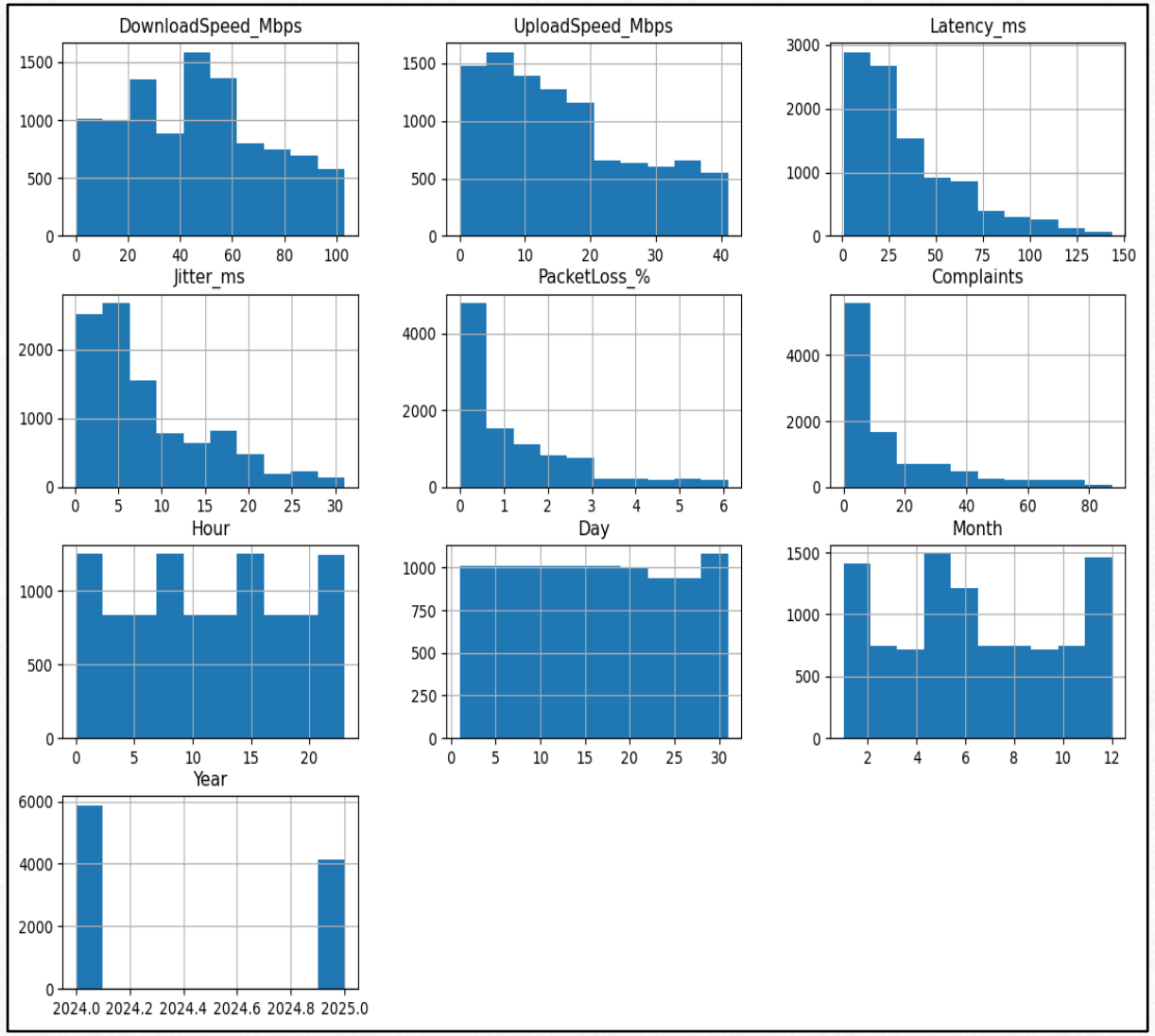
3) The Moderate_Downtime category appears with a moderate frequency.

4) The High_Downtime category has the lowest count among all.

5) This indicates that most observations experience low downtime, suggesting stable network performance in general.

6) 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.

Histograms: Distribution of Numerical Features



Conclusion:

1) The histograms display the distribution of all numerical variables in the dataset, such as network speed, latency, jitter, packet loss, and temporal features.

2) DownloadSpeed_Mbps and UploadSpeed_Mbps show moderately uniform distributions, indicating varying internet speeds among users.

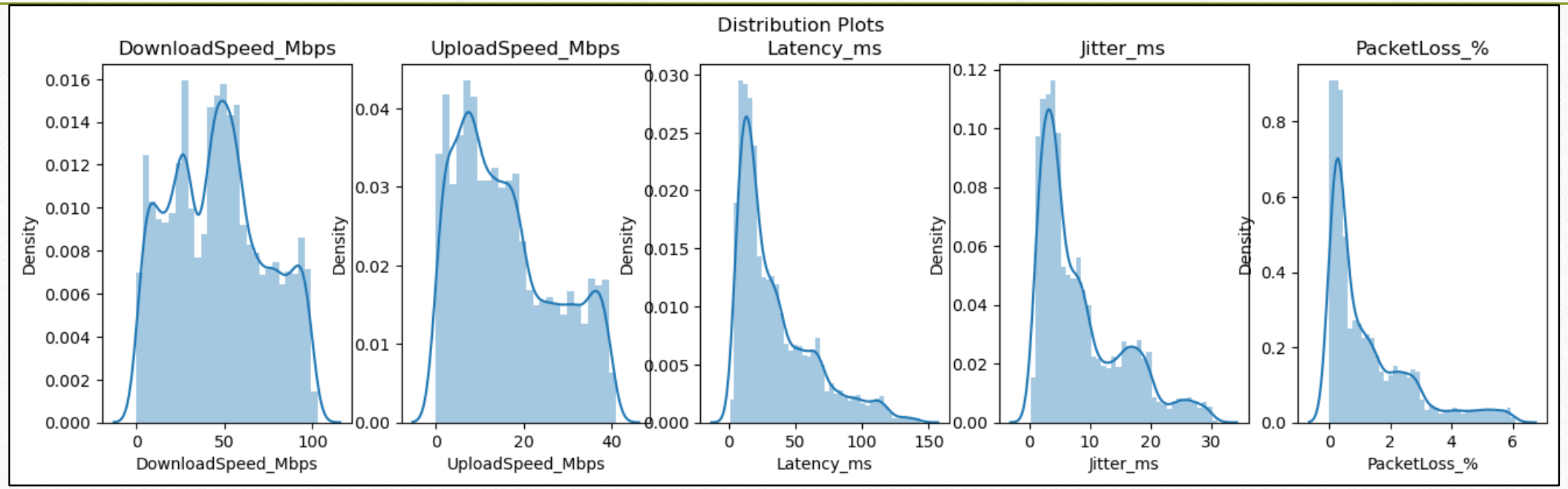
3) Latency_ms, Jitter_ms, PacketLoss_%, and Complaints are right-skewed, showing that most observations have low values while a few have very high values (possible outliers).

4) Hour, Day, and Month are evenly spread, suggesting data is well-distributed across different time periods.

5) The Year variable mainly covers 2024 and 2025, indicating data collected over two years.

6) 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.**

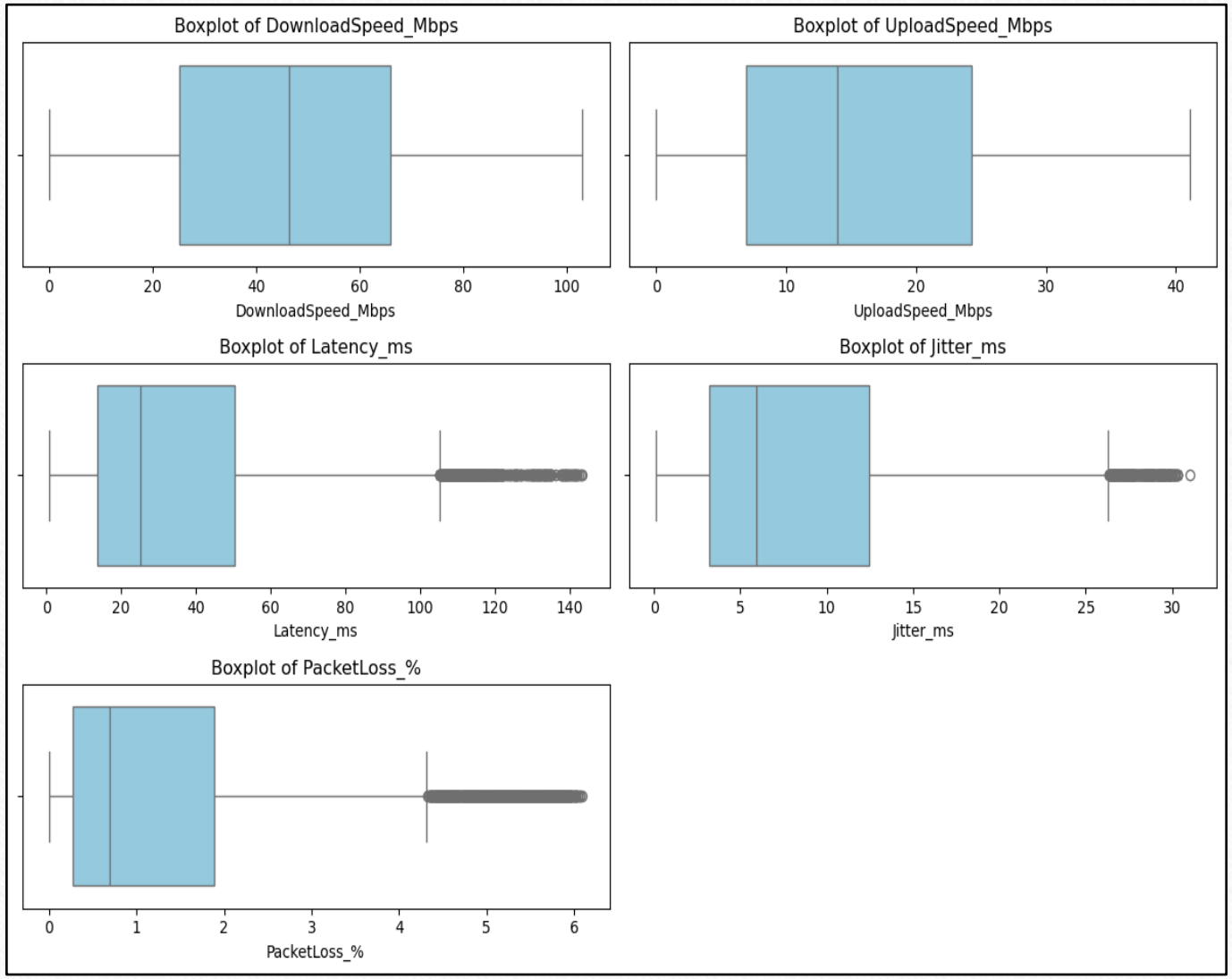
Distribution Plots: Network Performance Metrics



Conclusions:

1) The plots display the distribution of continuous variables such as DownloadSpeed_Mbps, UploadSpeed_Mbps, Latency_ms, Jitter_ms, and PacketLoss_%.
2) DownloadSpeed_Mbps and UploadSpeed_Mbps show a fairly bimodal distribution, indicating that users experience two distinct ranges of internet speeds.
3) 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.
4) The peaks near lower values in latency, jitter, and packet loss suggest that network conditions are generally stable for most users.
5) 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.

Boxplots: Network Performance Metrics



Conclusion:

1) 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_%.

2) DownloadSpeed_Mbps and UploadSpeed_Mbps show a fairly symmetric distribution with few or no extreme outliers, indicating stable internet speed values.

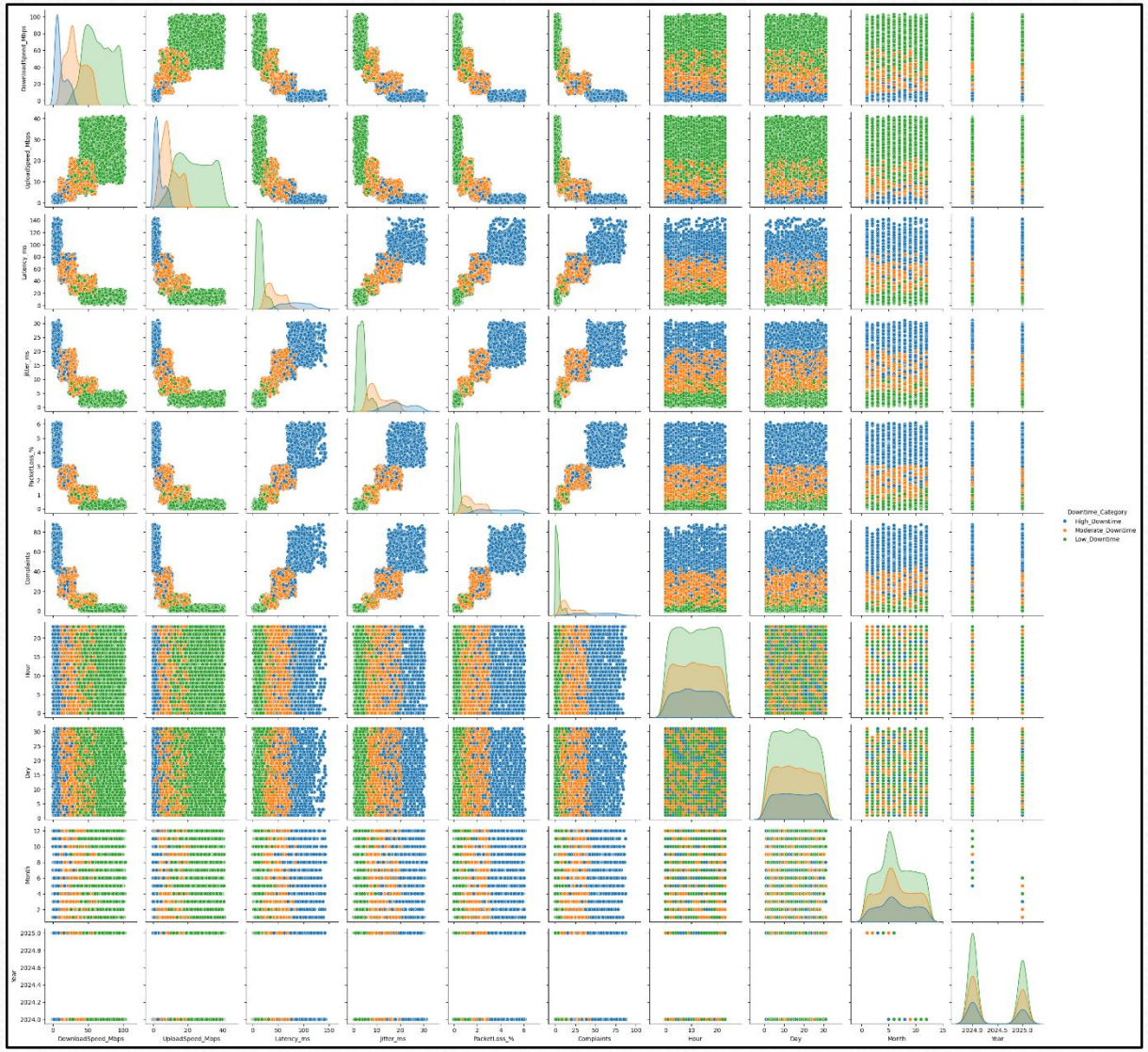
3) Latency_ms, Jitter_ms, and PacketLoss_% contain several outliers on the higher side, showing occasional spikes in delay, jitter, and packet loss.

4) The median values of these variables lie toward the lower end, suggesting that most users experience good network performance most of the time.

5) The presence of high-end outliers may correspond to rare instances of poor network conditions or high downtime events.

6) 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.

Features Pairplot



Conclusion:

1) 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*).

2) 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.

3) In contrast, High_Downtime points (red) are more concentrated in regions with low speeds and high latency/jitter, indicating poorer network performance.

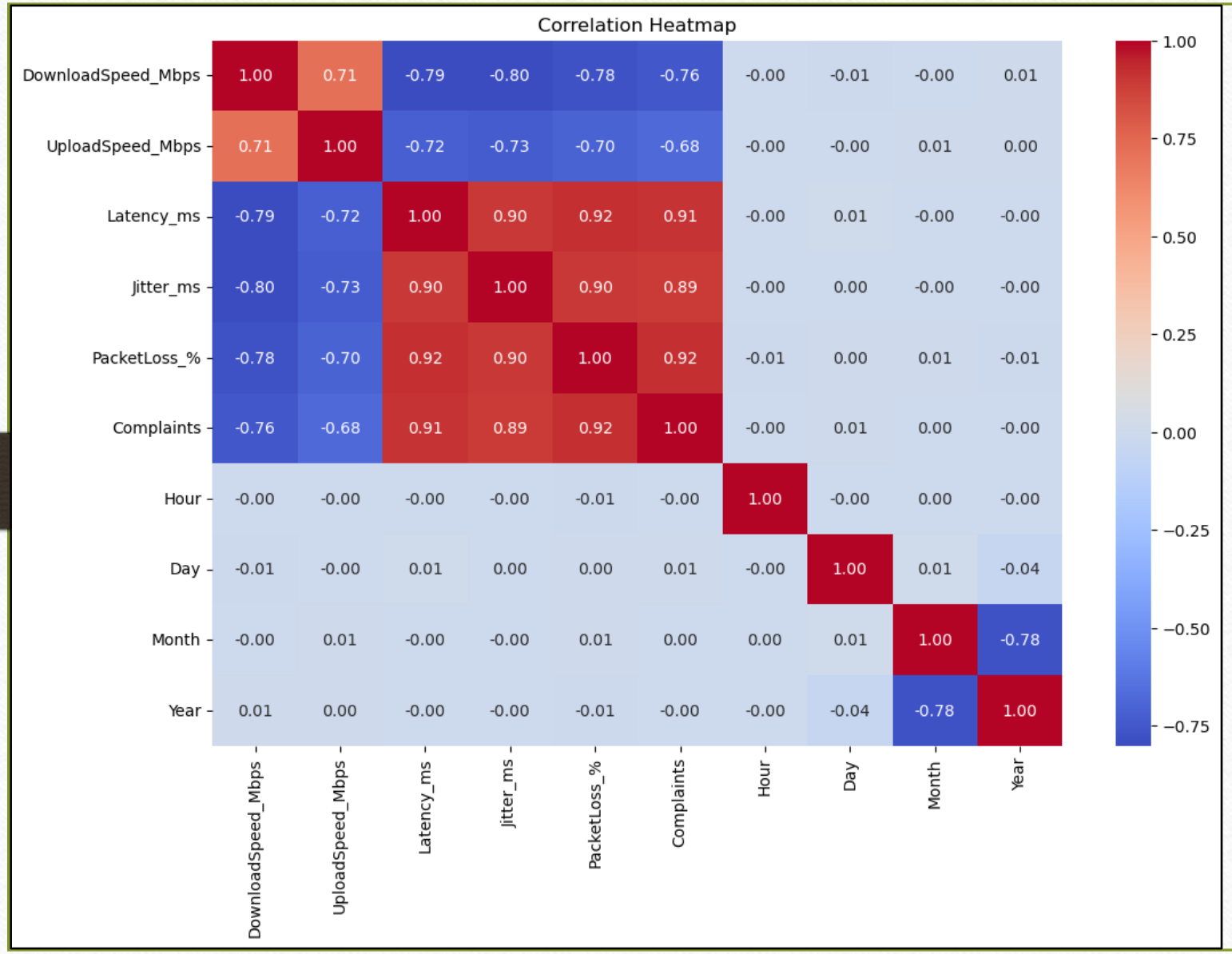
4) Moderate_Downtime points (orange) lie between the two, showing intermediate performance levels.

5) The plots confirm that network performance metrics are strongly related to downtime categories, supporting their importance as predictive features.

6) No strong linear relationships are visible among all features, suggesting that non-linear models (like Random Forest or XGBoost) may perform better.

7) Overall, the pairplot highlights clear separation patterns between classes, helping in understanding how performance metrics influence downtime classification.

Correlation Heatmap



Conclusion:

1) 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.

2) 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.

3) 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.

4) 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.

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

Conclusions:

1) The code removes outliers from the numerical columns (DownloadSpeed_Mbps, UploadSpeed_Mbps, Latency_ms, Jitter_ms, PacketLoss_%, Complaints) using the IQR method.
2) Values outside 1.5×IQR from Q1 and Q3 are filtered out to reduce extreme deviations.
3) After outlier removal, the dataset has 8,369 rows and 14 columns, ensuring cleaner data for modeling.
4) This helps improve model accuracy and robustness by reducing the effect of extreme values.

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
```

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

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

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

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

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'))
```

Preprocessor

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

```
ColumnTransformer(transformers=[('cat', OneHotEncoder(handle_unknown='ignore'),
                                 Index(['City', 'Locality', 'WeatherCondition'], dtype='object')),
                                ('num', StandardScaler(),
                                 Index(['DownloadSpeed_Mbps', 'UploadSpeed_Mbps', 'Latency_ms', 'Jitter_ms',
       'PacketLoss_%', 'Complaints'],
      dtype='object'))])
```

cat

```
Index(['City', 'Locality', 'WeatherCondition'], dtype='object')
```

OneHotEncoder

```
OneHotEncoder(handle_unknown='ignore')
```

num

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

StandardScaler

```
StandardScaler()
```

Conclusions:

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

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)
```

Define Models

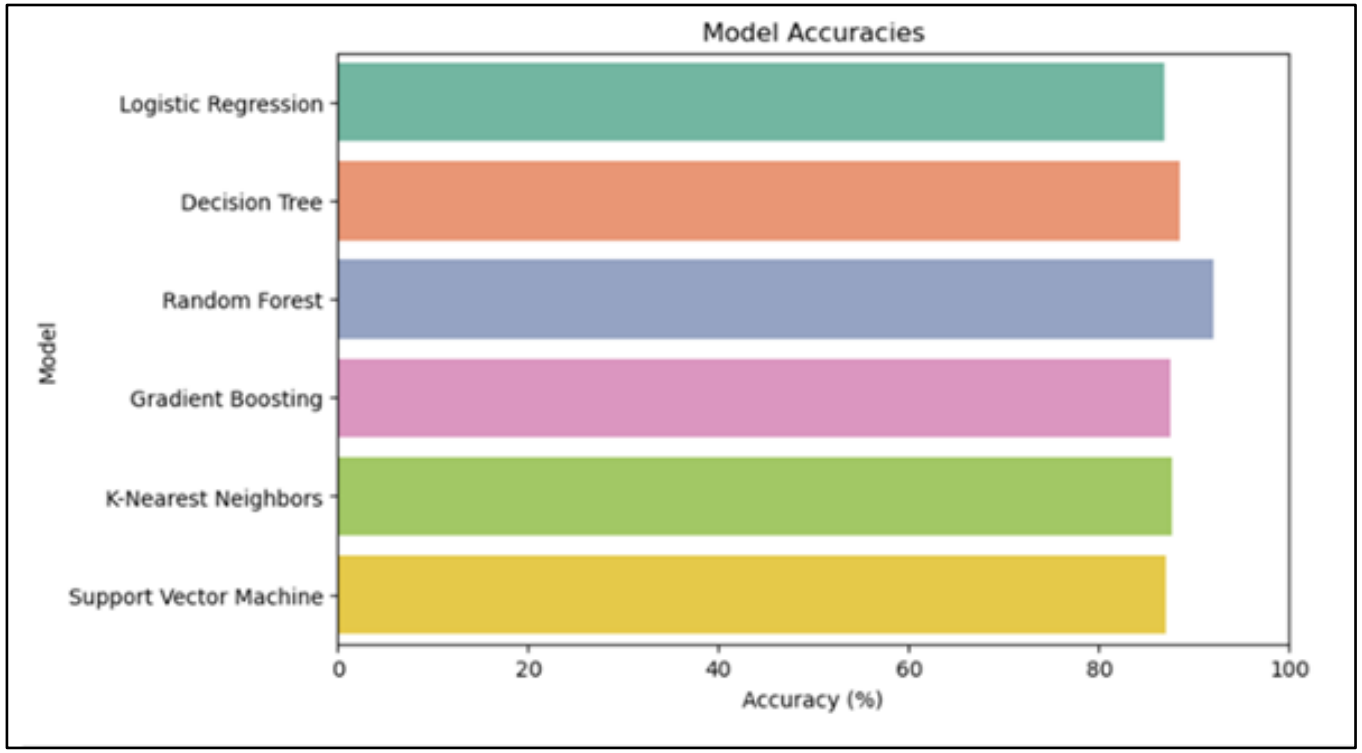
```python
# Define models
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'K-Nearest Neighbors': KNeighborsClassifier(),
 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

Evaluate Models

```python
# 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%
```

Visualize Model Accuracy



Testing

```
# 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
```

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']
```

```
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
```

requirements.txt

RESULT

Internet Downtime Prediction

Streamlit Web APP

Homepage

Internet Downtime Prediction

City

Select a City

Locality

Select a City first

Weather Condition

Select Weather Condition

Download Speed (Mbps) [Range: 0.0 – 200]

0.0

Upload Speed (Mbps) [Range: 0.0 – 50.0]

0.0

Latency (ms) [Range: 0.5 – 145.0]

0.5

Jitter (ms) [Range: 0.1 – 35.0]

0.1

Packet Loss (%) [Range: 0.0 – 7.0]

0.0

Complaints [Range: 0 – 100]

0

Predict Downtime

Internet Downtime Prediction

Sample 1

Output: Low_Downtime

Internet Downtime Prediction

City

Mumbai

Locality

Bandra

Weather Condition

Sunny

Download Speed (Mbps) [Range: 0.0 – 200]

150.0

Upload Speed (Mbps) [Range: 0.0 – 50.0]

45.0

Latency (ms) [Range: 0.5 – 145.0]

20.0

Jitter (ms) [Range: 0.1 – 35.0]

10.0

Packet Loss (%) [Range: 0.0 – 7.0]

1.0

Complaints [Range: 0 – 100]

10

Predict Downtime

Predicted Downtime Category: Low_Downtime

Classifier: Random Forest Classifier
Accuracy: 92.00 %
Built by: Suraj R. Yadav

Internet Downtime Prediction

Sample 2

Output: Moderate_Downtime

Internet Downtime Prediction

City

Chennai

Locality

Anna Nagar

Weather Condition

Stormy

Download Speed (Mbps) [Range: 0.0 – 200]

100.0

Upload Speed (Mbps) [Range: 0.0 – 50.0]

35.0

Latency (ms) [Range: 0.5 – 145.0]

100.0

Jitter (ms) [Range: 0.1 – 35.0]

20.0

Packet Loss (%) [Range: 0.0 – 7.0]

3.0

Complaints [Range: 0 – 100]

40

Predict Downtime

Predicted Downtime Category: Moderate_Downtime

Classifier: Random Forest Classifier
Accuracy: 92.00 %
Built by: Suraj R. Yadav

Internet Downtime Prediction

Sample 3

Output: High_Downtime

Internet Downtime Prediction

City

Delhi

Locality

Saket

Weather Condition

Rainy

Download Speed (Mbps) [Range: 0.0 – 200]

50.0

Upload Speed (Mbps) [Range: 0.0 – 50.0]

20.0

Latency (ms) [Range: 0.5 – 145.0]

120.0

Jitter (ms) [Range: 0.1 – 35.0]

30.0

Packet Loss (%) [Range: 0.0 – 7.0]

5.0

Complaints [Range: 0 – 100]

70

Predict Downtime

Predicted Downtime Category: High_Downtime

Classifier: Random Forest Classifier
Accuracy: 92.00 %
Built by: Suraj R. Yadav

Streamlit Web APP link : ***https://internet-downtime-prediction.streamlit.app/***

GitHub Project link : ***https://github.com/sryagit/Internet_Downtime_Prediction***

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.