Lab Report on

CSE 452 - Machine Learning Lab

Submitted To

Md. Ferdous

Lecturer
Dept. of CSE, BSMRSTU, Gopalganj
E-mail: md.ferdous@bsmrstu.edu.bd

Submitted By

Israt Jahan Reshma
ID:18CSE241
4th Year, 2nd Semester
Session: 2018-2019
Dept. of CSE, BSMRSTU
E-mail: israt.18cse241@bsmrstu.edu.bd

Department of Computer Science & Engineering
Bangabandhu Sheikh Mujibur Rahman Science & Technology University, Gopalganj – 8100

10 November 2024

ML Algorithms and Used dataset overview

1. **Data Analysis: IT Student Mental Survey Dataset (Kaggle)**
    a. **Algorithm**: Data Analysis
    b. **Purpose**: Analyze mental health challenges (stress, anxiety) faced by IT students.

2. **Linear Regression: Student Performance Factors Dataset (Kaggle)**
    a. **Algorithm**: Linear Regression
    b. **Purpose**: Predict student grades based on factors like demographics and study habits.

3. **Logistic Regression: Student Performance Dataset (Kaggle)**
    a. **Algorithm**: Logistic Regression
    b. **Purpose**: Predict pass/fail outcomes based on student characteristics.

4. **K Means Clustering: Daily Delhi Climate Dataset (Kaggle)**
    a. **Algorithm**: K Means Clustering
    b. **Purpose**: Categorize days into similar weather patterns (temperature, humidity, air quality).

5. **SVM: Hepatitis.csv Dataset (Kaggle)**
    a. **Algorithm**: Support Vector Machine (SVM)
    b. **Purpose**: Classify patients as having hepatitis or not based on medical data.

6. **ANN: Keras MNIST Dataset**
    a. **Algorithm**: Artificial Neural Network (ANN)
    b. **Purpose**: Classify handwritten digits (0-9) using image data.

7. **CNN: Cat and Dog Image Classification Dataset**
    a. **Algorithm**: Convolutional Neural Network (CNN)
    b. **Purpose**: Classify images as cat or dog.

8. **LSTM: DailyDelhiClimateTrain.csv and Test Dataset (Kaggle)**
    a. **Algorithm**: Long Short-Term Memory (LSTM)
    b. **Purpose**: Predict future climate trends based on historical data.

Data Analysis

The Data set is taken from Kaggle named as IT student Mental Servay.

In [46]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [47]:
```
df = pd.read_csv('MentalHealthSurveyfoITstudents.csv');
df.head()
```

Out[47]:

	gender	age	university	degree_level	degree_major	academic_year	cgpa	residential_status	can
0	Male	20	PU	Undergraduate	Data Science	2nd year	3.0-3.5	Off-Campus	
1	Male	20	UET	Postgraduate	Computer Science	3rd year	3.0-3.5	Off-Campus	
2	Male	20	FAST	Undergraduate	Computer Science	3rd year	2.5-3.0	Off-Campus	
3	Male	20	UET	Undergraduate	Computer Science	3rd year	2.5-3.0	On-Campus	
4	Female	20	UET	Undergraduate	Computer Science	3rd year	3.0-3.5	Off-Campus	

5 rows × 21 columns

In [48]:
```
print("Shape : ", df.shape)
```

```
Shape :  (87, 21)
```

In [49]:
```
df.duplicated().sum()
df.isna().sum().sum()
```

Out[49]:
```
0
```

In [50]:
```
df.describe().T
```

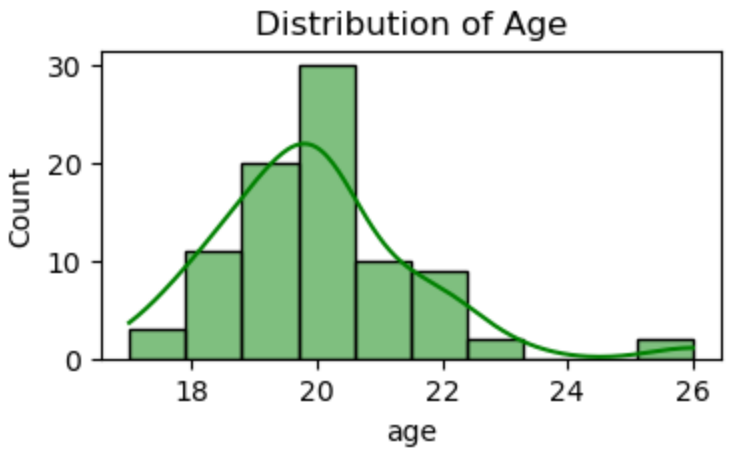
Out[50]:

	count	mean	std	min	25%	50%	75%	max
age	87.0	19.942529	1.623636	17.0	19.0	20.0	21.0	26.0
study_satisfaction	87.0	3.931034	1.043174	1.0	3.0	4.0	5.0	5.0
academic_workload	87.0	3.885057	0.854880	2.0	3.0	4.0	4.5	5.0
academic_pressure	87.0	3.781609	1.125035	1.0	3.0	4.0	5.0	5.0
financial_concerns	87.0	3.390805	1.400634	1.0	2.5	3.0	5.0	5.0
social_relationships	87.0	2.781609	1.175578	1.0	2.0	3.0	4.0	5.0
depression	87.0	3.218391	1.367609	1.0	2.0	3.0	4.0	5.0
anxiety	87.0	3.218391	1.297809	1.0	2.0	3.0	4.0	5.0
isolation	87.0	3.241379	1.405682	1.0	2.0	3.0	4.5	5.0
future_insecurity	87.0	3.011494	1.385089	1.0	2.0	3.0	4.0	5.0

In [51]:
```
df['gender'].value_counts().T
```

Out[51]:
```
gender
Male      63
Female    24
Name: count, dtype: int64
```

In [52]:
```
plt.figure(figsize=(4,2))
sns.histplot(df['age'],kde=True,color='green',edgecolor="black")
plt.title("Distribution of Age")
plt.xlabel("age")
plt.show()
```



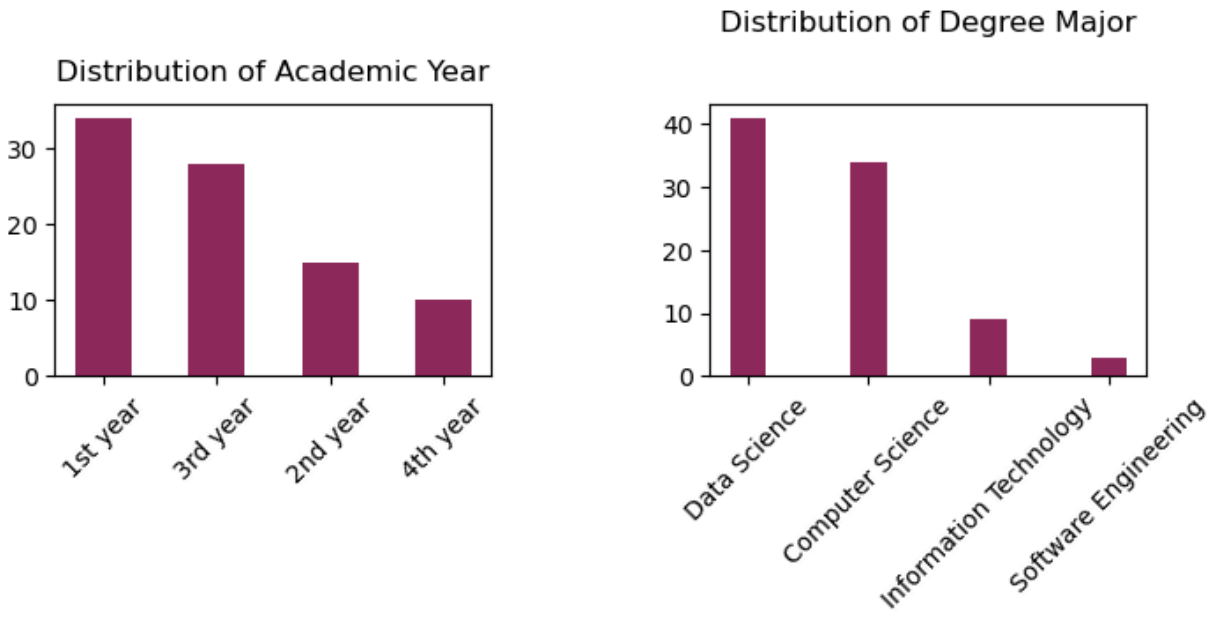
In [53]:
```
df['degree_level'].value_counts().T
```

Out[53]:
```
degree_level
Undergraduate    85
Postgraduate      2
Name: count, dtype: int64
```

In [54]:
```
df['degree_major'].value_counts().T
```

Out[54]:
```
degree_major
Data Science                41
Computer Science            34
Information Technology       9
Software Engineering         3
Name: count, dtype: int64
```

In [56]:
```
fig, axs = plt.subplots(1, 2, figsize=(8, 2))

axs[0].bar(df['academic_year'].value_counts().index,
           df['academic_year'].value_counts().values,
           color="#8f285b", width=0.5)
axs[0].set_title("Distribution of Academic Year", pad=10)
axs[0].tick_params(axis='x', rotation=45)

axs[1].bar(df['degree_major'].value_counts().index,
           df['degree_major'].value_counts().values,
           color="#8f285b", width=0.3)
axs[1].set_title("Distribution of Degree Major", pad=30)
axs[1].tick_params(axis='x', rotation=45)

plt.subplots_adjust(wspace=0.5)
plt.show()
```



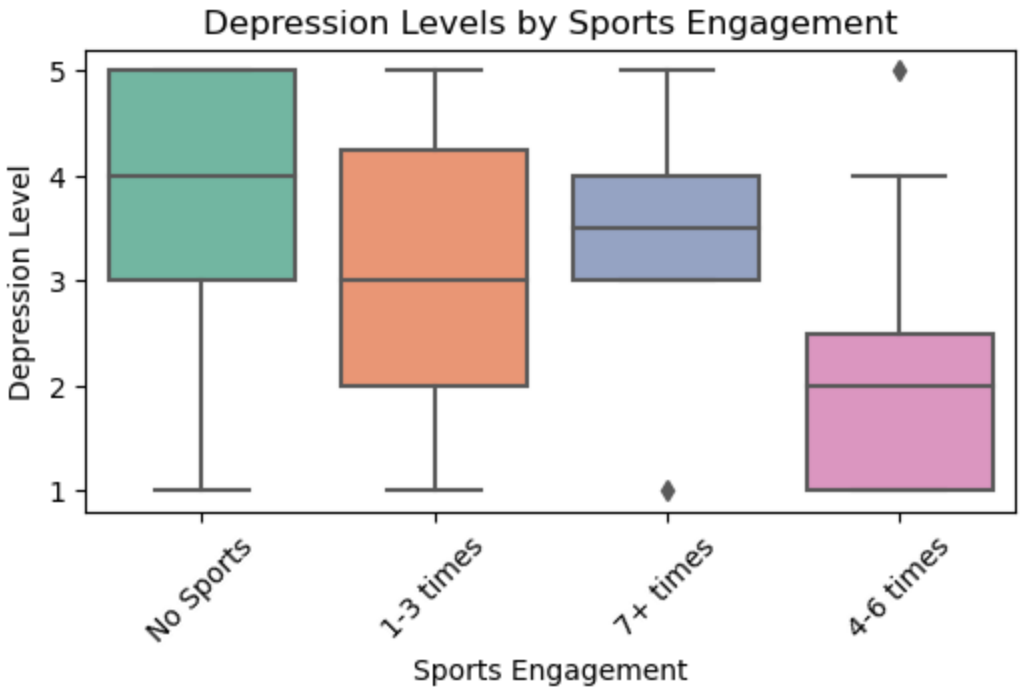
In [57]:
```
df.cgpa.value_counts().T
```

Out[57]:
```
cgpa
3.0-3.5    27
3.5-4.0    27
2.5-3.0    22
0.0-0.0     5
2.0-2.5     4
1.5-2.0     2
Name: count, dtype: int64
```

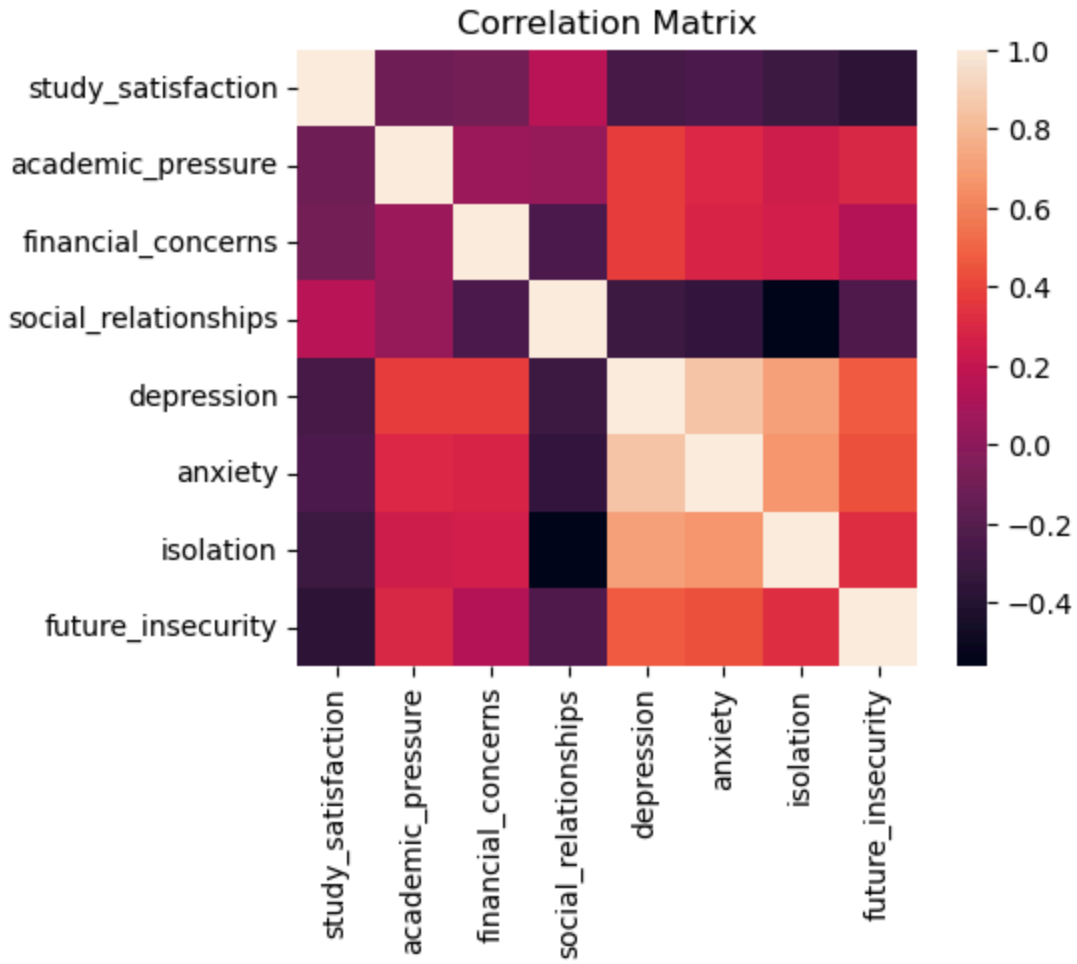
In [58]:
```
df.residential_status.value_counts().T
```

Out[58]:
```
residential_status
Off-Campus    65
On-Campus     22
Name: count, dtype: int64
```

In [59]:
```
plt.figure(figsize=(6, 3))
sns.boxplot(x='sports_engagement', y='depression', data=df, palette="Set2")
plt.title('Depression Levels by Sports Engagement')
plt.xlabel('Sports Engagement')
plt.ylabel('Depression Level')
plt.xticks(rotation=45)
plt.show()
```



In [60]:
```
data_corr=df[['study_satisfaction','academic_pressure','financial_concerns','social_re
#data_corr
plt.figure(figsize=(5, 4))
sns.heatmap(data_corr)
plt.title('Correlation Matrix')
plt.show()
```



Linear Regression

Take Student Performance Factors Dataset from Kaggle

In [66]:
```
df = pd.read_csv('StudentPerformanceFactors.csv')
df.head(2)
```

Out[66]:

	Hours_Studied	Attendance	Parental_Involvement	Access_to_Resources	Extracurricular_Activities	Sl
0	23	84	Low	High	No	
1	19	64	Low	Medium	No	

In [67]:
```
df=df.drop_duplicates()
#print(df.isna().sum())
df=df.dropna()
```

In [68]:
```
df.describe()
```

Out[68]:

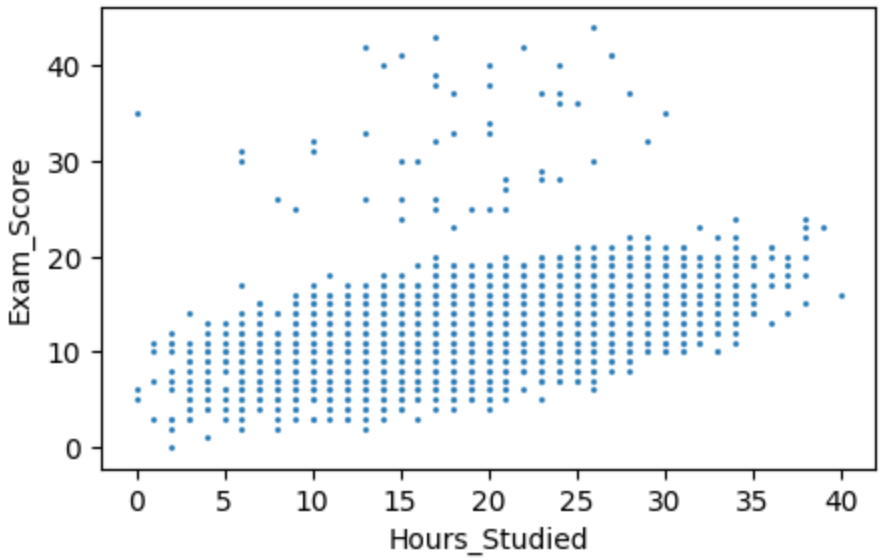
	Hours_Studied	Attendance	Sleep_Hours	Previous_Scores	Tutoring_Sessions	Physical_Activity
count	6378.000000	6378.000000	6378.000000	6378.000000	6378.000000	6378.000000
mean	19.977109	80.020853	7.034964	75.066165	1.495296	2.972719
std	5.985460	11.550723	1.468033	14.400389	1.233984	1.028926
min	1.000000	60.000000	4.000000	50.000000	0.000000	0.000000
25%	16.000000	70.000000	6.000000	63.000000	1.000000	2.000000
50%	20.000000	80.000000	7.000000	75.000000	1.000000	3.000000
75%	24.000000	90.000000	8.000000	88.000000	2.000000	4.000000
max	44.000000	100.000000	10.000000	100.000000	8.000000	6.000000

In [69]:
```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
for i in df.columns:
    df[i] = label_encoder.fit_transform(df[i])
df.head(2)
```

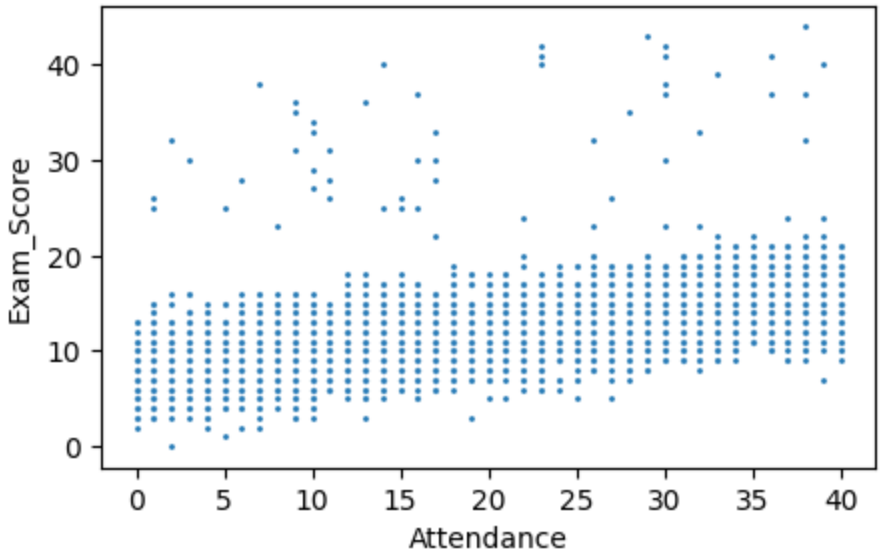
Out[69]:

	Hours_Studied	Attendance	Parental_Involvement	Access_to_Resources	Extracurricular_Activities	Sl
0	22	24	1	0	0	
1	18	4	1	2	0	

In [97]:
```
plt.figure(figsize=(5, 3))  # Adjust the figure size as needed
sns.scatterplot(data=df, x='Hours_Studied', y='Exam_Score', s=5)
plt.show()
```



In [98]:
```
plt.figure(figsize=(5, 3))
sns.scatterplot(data=df, x='Attendance', y='Exam_Score', s=5)
plt.show()
```

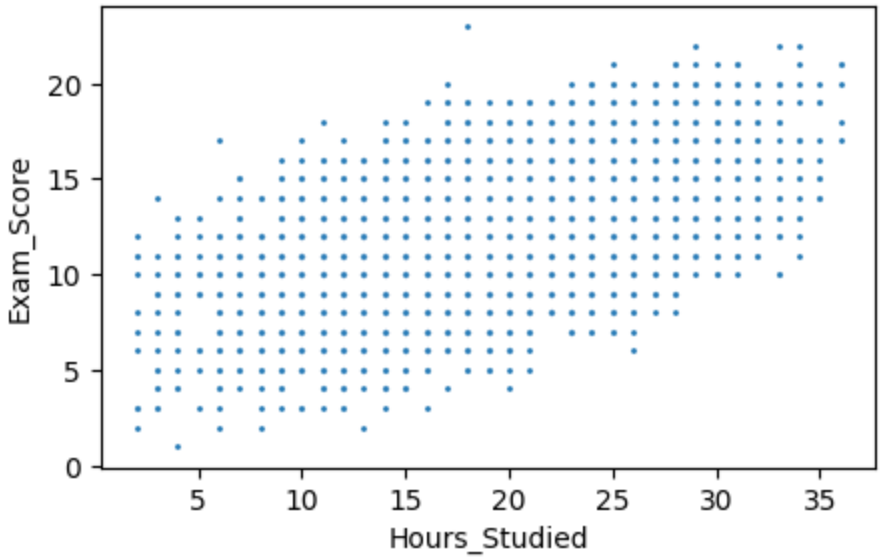


In [99]:
```
#removing outliers
mean = np.mean(df, axis=0)
std_dev = np.std(df, axis=0)
z_scores = (df - mean) / std_dev
outliers = np.abs(z_scores) > 3
df = df[~np.any(outliers, axis=1)]
df.head()
```

Out[99]:

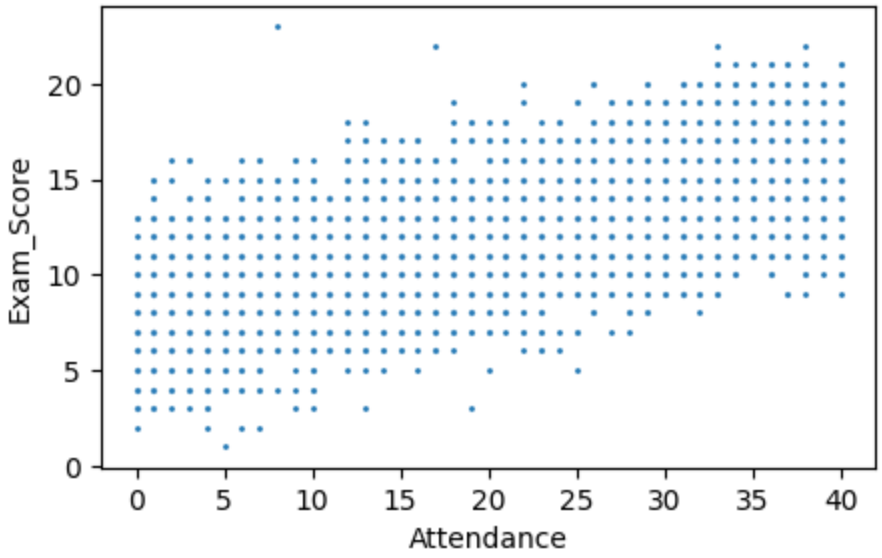
	Hours_Studied	Attendance	Parental_Involvement	Access_to_Resources	Extracurricular_Activities	Sl
0	22	24	1	0	0	
1	18	4	1	2	0	
2	23	38	2	2	1	
3	28	29	1	2	1	
4	18	32	2	2	1	

In [100…
```
plt.figure(figsize=(5, 3))
sns.scatterplot(data=df, x='Hours_Studied', y='Exam_Score',s=5)
plt.show()
```



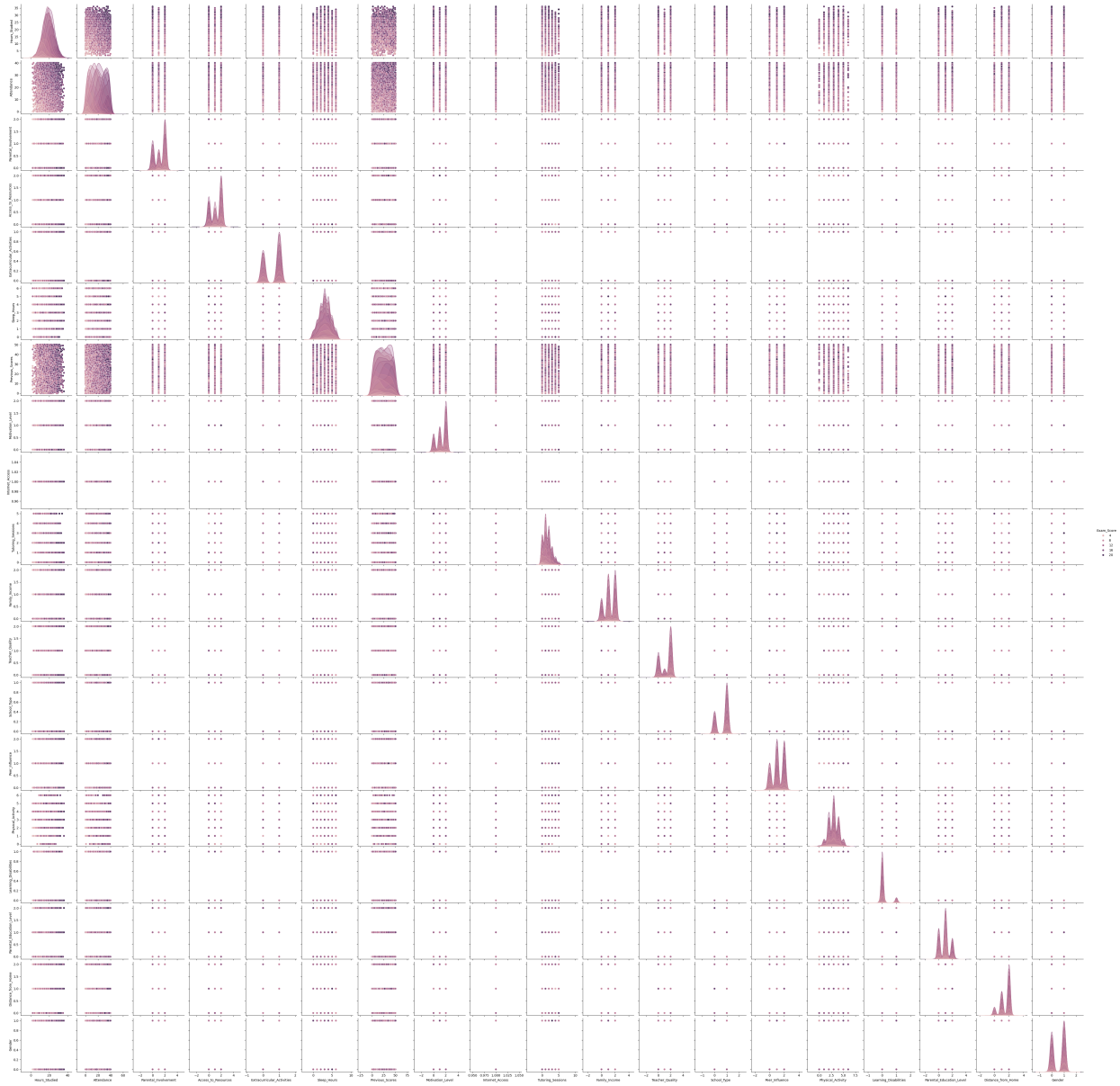
In [101…

```
plt.figure(figsize=(5, 3))
sns.scatterplot(data=df, x='Attendance', y='Exam_Score',s=5)
plt.show()
```



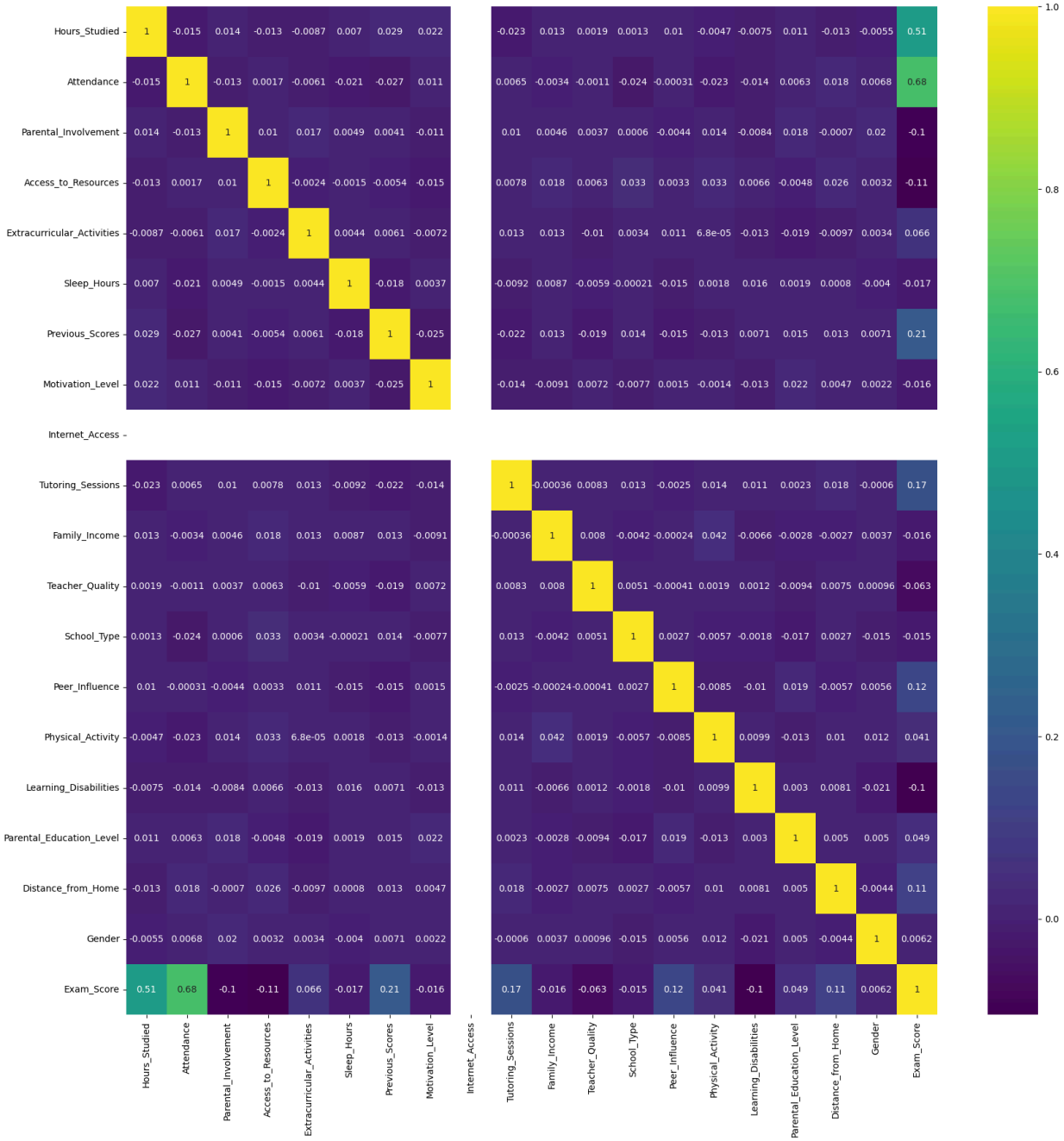
In [102…

```
sns.pairplot(df,hue="Exam_Score")
plt.show()
```



In [103…

```
Correlation_Matrix=df.corr()
plt.figure(figsize=(20,20))
sns.heatmap(Correlation_Matrix,annot=True,cmap='viridis')
plt.show()
```



In [104…
```
X=df[['Hours_Studied','Attendance']]
y=df['Exam_Score']
```

In [105…
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

In [106…
```
from sklearn.linear_model import LinearRegression
reg=LinearRegression()
```

In [107…
```
reg.fit(x_train,y_train)
```

Out[107]:
```
LinearRegression()
```

In [108…
```
y_pred=reg.predict(x_test)
```

In [109…
```
from sklearn.metrics import r2_score
r2_score(y_test,y_pred)
```

Out[109]:
```
0.7263622686013315
```

Logistic Regression

take student performance data set from kaggle

In [110…
```
df = pd.read_csv('student performance.csv')
```

In [111…
```
df.head()
```

Out[111]:

	Marital status	Application mode	Application order	Course	Daytime/evening attendance	Previous qualification	Nacionality	Mother's qualificatior
0	1	8	5	2	1	1	1	13
1	1	6	1	11	1	1	1	1
2	1	1	5	5	1	1	1	22
3	1	8	2	15	1	1	1	23
4	2	12	1	3	0	1	1	22

5 rows × 35 columns

In [112…
```
df.isna().sum().sum()
```

Out[112]:
```
0
```

In [113…
```
df['Target'].value_counts()
```

Out[113]:

Target	count
Graduate	2209
Dropout	1421
Enrolled	794

dtype: int64

In [114…
```
df['Target']=df['Target'].apply(lambda x: 1 if x=='Graduate' else 0)
df['Target'].value_counts()
```

Out[114]:

Target	count
0	2215
1	2209

dtype: int64

In [115…
```
X=df.drop('Target', axis=1)
y=df['Target']
```

In [116…
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
X_train, X_test, Y_train, Y_test= train_test_split(X, y, test_size=0.2, random_state=4
logmodel=LogisticRegression(max_iter=5000)
logmodel.fit(X_train, Y_train)
```

Out[116]:

LogisticRegression

LogisticRegression(max_iter=5000)

In [117…
```
predictions=logmodel.predict(X_test)
```

In [118…
```
from sklearn.metrics import classification_report
print(classification_report(Y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.89      0.79      0.84       467
           1       0.79      0.89      0.84       418

    accuracy                           0.84       885
   macro avg       0.84      0.84      0.84       885
weighted avg       0.84      0.84      0.84       885
```

K means Clustering

Data set is taken from Kaggle : name: Daily Delhi Climate dataset

In [101…
```
df=pd.read_csv("DailyDelhiClimateTrain.csv")
df.head()
```

Out[101]:

	date	meantemp	humidity	wind_speed	meanpressure
0	2013-01-01	10.000000	84.500000	0.000000	1015.666667
1	2013-01-02	7.400000	92.000000	2.980000	1017.800000
2	2013-01-03	7.166667	87.000000	4.633333	1018.666667
3	2013-01-04	8.666667	71.333333	1.233333	1017.166667
4	2013-01-05	6.000000	86.833333	3.700000	1016.500000

In [102…
```
df.shape
```

Out[102]:
```
(1462, 5)
```

In [103…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1462 entries, 0 to 1461
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   date          1462 non-null   object 
 1   meantemp      1462 non-null   float64
 2   humidity      1462 non-null   float64
 3   wind_speed    1462 non-null   float64
 4   meanpressure  1462 non-null   float64
dtypes: float64(4), object(1)
memory usage: 57.2+ KB
```

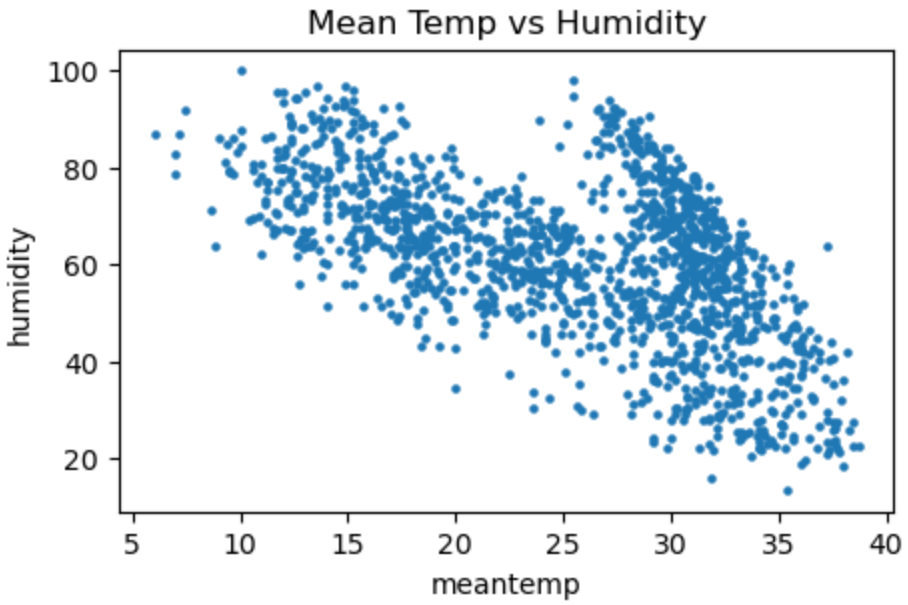
In [104…
```
df.describe()
```

Out[104]:

	meantemp	humidity	wind_speed	meanpressure
count	1462.000000	1462.000000	1462.000000	1462.000000
mean	25.495521	60.771702	6.802209	1011.104548
std	7.348103	16.769652	4.561602	180.231668
min	6.000000	13.428571	0.000000	-3.041667
25%	18.857143	50.375000	3.475000	1001.580357
50%	27.714286	62.625000	6.221667	1008.563492
75%	31.305804	72.218750	9.238235	1014.944901
max	38.714286	100.000000	42.220000	7679.333333

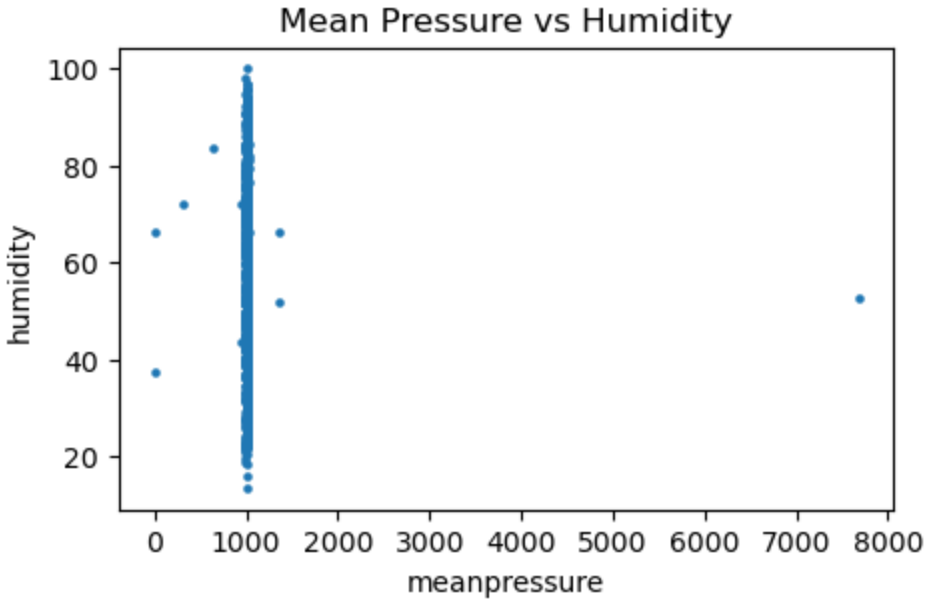
In [113…
```
df.plot.scatter(x='meantemp', y='humidity', title='Mean Temp vs Humidity', s=5, figsiz
```

Out[113]:
```
<Axes: title={'center': 'Mean Temp vs Humidity'}, xlabel='meantemp', ylabel='humidit
y'>
```



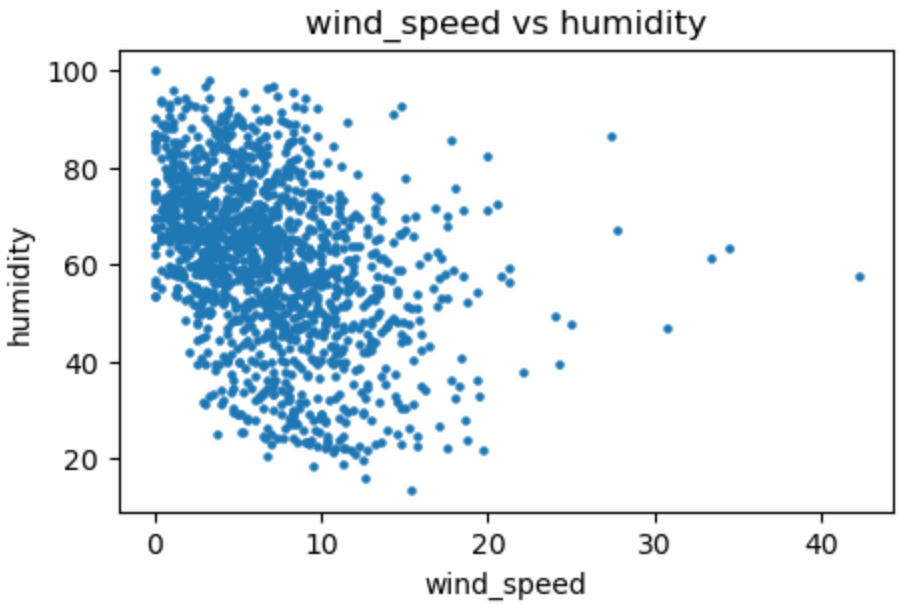
In [112…
```
df.plot.scatter(x='meanpressure', y='humidity', title='Mean Pressure vs Humidity', s=5
```

Out[112]:
```
<Axes: title={'center': 'Mean Pressure vs Humidity'}, xlabel='meanpressure', ylabel
='humidity'>
```



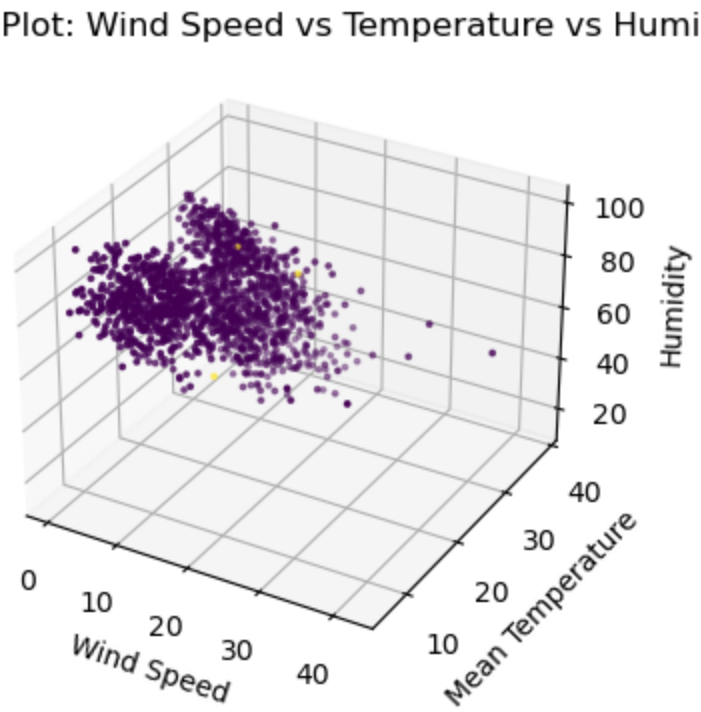
In [114…
```
df.plot(x='wind_speed',
        y='humidity',
        kind='scatter',
        title='wind_speed vs humidity',s=5, figsize=(5, 3))
```

Out[114]:
```
<Axes: title={'center': 'wind_speed vs humidity'}, xlabel='wind_speed', ylabel='humid
ity'>
```



In [115…
```
from sklearn.cluster import KMeans
import plotly.express as px

x= df[['humidity','meanpressure','meantemp','wind_speed']]
km = KMeans(n_clusters=3, n_init=10)
df['cluster']= km.fit_predict(x)
df['cluster']=df['cluster'] \
    .astype('category')
```

In [118…
```
fig = plt.figure(figsize=(6, 4))
ax = fig.add_subplot(111, projection='3d')

# Plot the 3D scatter plot
ax.scatter(df['wind_speed'], df['meantemp'], df['humidity'], c=df['cluster'].astype('c

# Set the labels and titleax.set_xlabel('Wind Speed')
ax.set_ylabel('Mean Temperature')
ax.set_zlabel('Humidity')
ax.set_title('3D Scatter Plot: Wind Speed vs Temperature vs Humidity')

# Show the plot
plt.show()
```



In [117…
```
df.drop(columns=['date'],inplace=True)
```

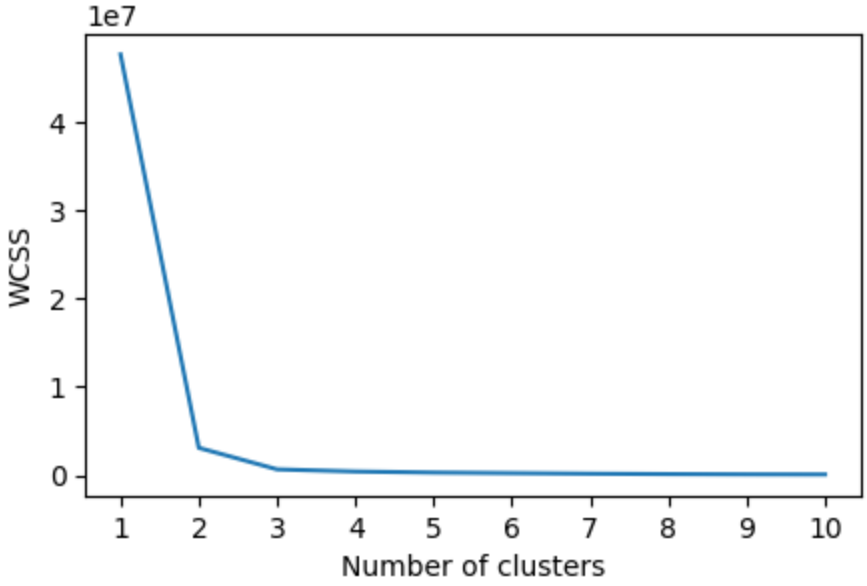
In [32]:
```
x=df.drop(columns=['wind_speed'])
y=df['wind_speed']
```

In [35]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

In [36]:
```
wcss = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, init='k-means++', n_init=10)
    kmeans.fit(x_train)
    wcss.append(kmeans.inertia_)
wcss
```

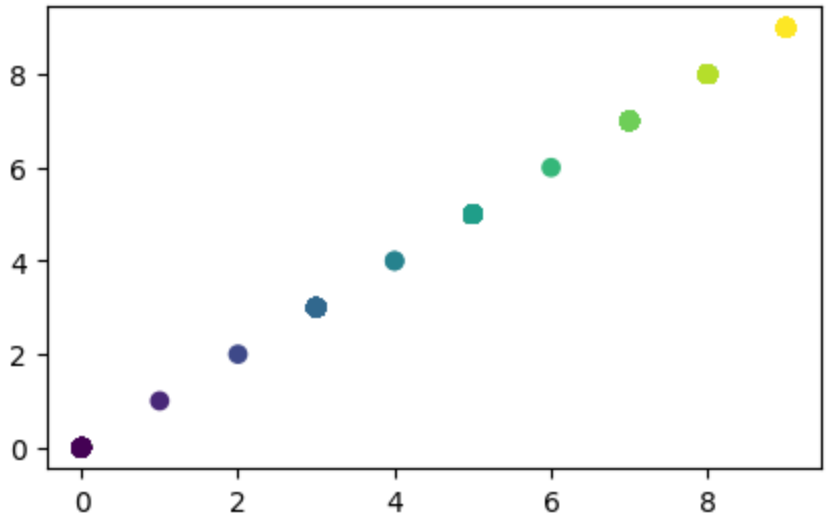
Out[36]:
```
[47572716.82843734,
 3078058.3838992873,
 644083.7667092744,
 400171.0454776609,
 284528.8565589274,
 216098.55890250346,
 153542.54396055674,
 109239.52061460483,
 82544.78736091012,
 70091.01770029642]
```

In [38]:
```
plt.figure(figsize=(5, 3))  # Specify smaller width and height in inches
plt.plot(range(1, 11), wcss)
plt.xticks(range(1, 11))
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')  # Add ylabel if needed
plt.show()
```



In [39]:
```
x_pred=kmeans.fit_predict(x_train)
```

In [40]:
```
plt.figure(figsize=(5, 3))  # Specify the width and height in inches
plt.scatter(x_pred, x_pred, c=x_pred)
plt.show()
```



In [47]:
```
from kneed import KneeLocator
k1= KneeLocator(range(1,11),wcss,curve='convex',direction='decreasing')
k1.elbow
```

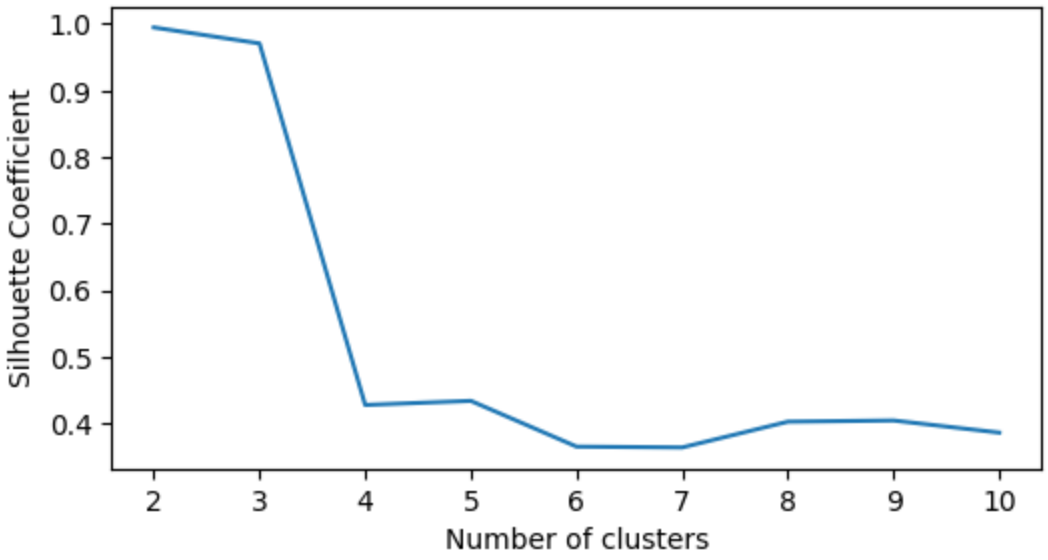
Out[47]:
```
2
```

In [48]:
```
#performance metrics
from sklearn.metrics import silhouette_score
silhouette_coefficients=[]
for k in range(2,11):
    kmeans= KMeans(n_clusters=k,init="k-means++",n_init=10)
    kmeans.fit(x_train)
    score=silhouette_score(x_train,kmeans.labels_)
    silhouette_coefficients.append(score)
```

In [49]:
```
silhouette_coefficients
```

Out[49]:
```
[0.9947409709877785,
 0.9707440447233446,
 0.4274533839989495,
 0.4336048534773166,
 0.36487063247390333,
 0.3636351197961745,
 0.4023410404344692,
 0.404124154769077,
 0.3859009921674613]
```

In [50]:
```
plt.figure(figsize=(6, 3))  # Adjust the width and height as needed
plt.plot(range(2, 11), silhouette_coefficients)
plt.xticks(range(2, 11))
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Coefficient')
plt.show()
```



SVM

use "hepatitis.csv" from Kaggle

In [5]:
```
## Read "hepatitis.csv" using pandas
# target =  1: Die; 2: Live
data = pd.read_csv("hepatitis.csv", na_values="?")
data.head()
```

Out[5]:

	class	age	sex	steroid	antivirals	fatigue	malaise	anorexia	liver_big	liver_firm	spleen_palable
0	2	30	2	1	2	2	2	2	1	2	2
1	2	50	1	1	2	1	2	2	1	2	2
2	2	78	1	2	2	1	2	2	2	2	2
3	2	34	1	2	2	2	2	2	2	2	2
4	2	34	1	2	2	2	2	2	2	2	2

In [6]:
```
data.shape
```

Out[6]:
```
(142, 20)
```

In [7]:
```
data.describe()
```

Out[7]:

	class	age	sex	steroid	antivirals	fatigue	malaise	anorexia
count	142.000000	142.000000	142.000000	142.000000	142.000000	142.000000	142.000000	142.000000
mean	1.816901	40.816901	1.105634	1.514085	1.838028	1.359155	1.619718	1.795775
std	0.388116	12.189182	0.308456	0.501571	0.369729	0.481451	0.487174	0.404561
min	1.000000	7.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	2.000000	32.000000	1.000000	1.000000	2.000000	1.000000	1.000000	2.000000
50%	2.000000	39.000000	1.000000	2.000000	2.000000	1.000000	2.000000	2.000000
75%	2.000000	50.000000	1.000000	2.000000	2.000000	2.000000	2.000000	2.000000
max	2.000000	78.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000

In [8]:
```
data['class'].value_counts()
```

Out[8]:

class	count
2	116
1	26

dtype: int64

In [9]:
```
cat_cols = data.columns[data.nunique() < 5]
```

In [10]:
```
num_cols = data.columns[data.nunique() >= 5]
```

In [11]:
```
num_cols = ["age", "bilirubin","alk_phosphate", "albumin", "protime"]
cat_cols = ['sex', 'steroid', 'antivirals', 'fatigue', 'malaise', 'anorexia', 'liver_b
```

In [12]:
```
data.isna().sum().sum()
```

Out[12]:
```
0
```

In [13]:
```
X = data.drop(['class'],axis = 1)
y = data['class']
print(X.shape, y.shape)
```

```
(142, 19) (142,)
```

In [15]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,random_state
```

In [16]:
```
X_train[cat_cols] = X_train[cat_cols].astype('int')

# Test
X_test[cat_cols] = X_test[cat_cols].astype('int')
```

In [17]:
```
X_train = pd.get_dummies(X_train, columns=cat_cols, drop_first=True)# Train
X_test = pd.get_dummies(X_test, columns=cat_cols, drop_first=True)# Test
```

In [18]:
```
X_train.columns
```

Out[18]:
```
Index(['age', 'bilirubin', 'alk_phosphate', 'sgot', 'albumin', 'protime',
       'sex_2', 'steroid_2', 'antivirals_2', 'fatigue_2', 'malaise_2',
       'anorexia_2', 'liver_big_2', 'liver_firm_2', 'spleen_palable_2',
       'spiders_2', 'ascites_2', 'varices_2', 'histology_2'],
      dtype='object')
```

In [19]:
```
from sklearn.preprocessing import StandardScaler
```

In [21]:
```
from sklearn.preprocessing import StandardScaler

# Ensure the columns in num_cols are of dtype float64 for compatibility with scaler tr
X_train.loc[:, num_cols] = X_train.loc[:, num_cols].astype('float64')
X_test.loc[:, num_cols] = X_test.loc[:, num_cols].astype('float64')

# Initialize and fit the scaler
scaler = StandardScaler()
scaler.fit(X_train.loc[:, num_cols])

# Apply the transformation
X_train.loc[:, num_cols] = scaler.transform(X_train.loc[:, num_cols])
X_test.loc[:, num_cols] = scaler.transform(X_test.loc[:, num_cols])
```

MODEL BUILDING - SVM

In [22]:
```
from sklearn.svm import SVC
linear_svm = SVC(kernel='linear', C=1, random_state=0)
```

In [23]:
```
# Train the classifier
linear_svm.fit(X=X_train, y= y_train)
```

Out[23]:

SVC

SVC(C=1, kernel='linear', random_state=0)

In [24]:

```
## Predict
train_predictions = linear_svm.predict(X_train)
test_predictions = linear_svm.predict(X_test)

### Train data accuracy
from sklearn.metrics import accuracy_score,f1_score,confusion_matrix

print("TRAIN Conf Matrix : \n", confusion_matrix(y_train, train_predictions))
print("\nTRAIN DATA ACCURACY",accuracy_score(y_train,train_predictions))
print("\nTrain data f1-score for class '1'",f1_score(y_train,train_predictions,pos_lab
print("\nTrain data f1-score for class '2'",f1_score(y_train,train_predictions,pos_lab

### Test data accuracy
print("\n\n-------------------------------------\n\n")

print("TEST Conf Matrix : \n", confusion_matrix(y_test, test_predictions))
print("\nTEST DATA ACCURACY",accuracy_score(y_test,test_predictions))
print("\nTest data f1-score for class '1'",f1_score(y_test,test_predictions,pos_label=
print("\nTest data f1-score for class '2'",f1_score(y_test,test_predictions,pos_label=
```

```
TRAIN Conf Matrix :
 [[13  7]
 [ 3 90]]

TRAIN DATA ACCURACY 0.911504424778761

Train data f1-score for class '1' 0.7222222222222222

Train data f1-score for class '2' 0.9473684210526315


-------------------------------------


TEST Conf Matrix :
 [[ 1  5]
 [ 2 21]]

TEST DATA ACCURACY 0.7586206896551724

Test data f1-score for class '1' 0.2222222222222222

Test data f1-score for class '2' 0.8571428571428571
```

Kernel Trick:

In [25]:

```
svc = SVC(kernel='rbf', random_state=0, gamma=0.01, C=1)
svc
```

Out[25]:

SVC

SVC(C=1, gamma=0.01, random_state=0)

In [26]:
```
svc.fit(X=X_train, y= y_train)
```

Out[26]:
```
SVC(C=1, gamma=0.01, random_state=0)
```

In [27]:
```
## Predicttrain_predictions = svc.predict(X_train)
test_predictions = svc.predict(X_test)

### Train data accuracy
print("TRAIN Conf Matrix : \n", confusion_matrix(y_train, train_predictions))
print("\nTRAIN DATA ACCURACY",accuracy_score(y_train,train_predictions))
print("\nTrain data f1-score for class '1'",f1_score(y_train,train_predictions,pos_lab
print("\nTrain data f1-score for class '2'",f1_score(y_train,train_predictions,pos_lab

### Test data accuracy
print("\n\n---------------------------------------\n\n")

print("TEST Conf Matrix : \n", confusion_matrix(y_test, test_predictions))
print("\nTEST DATA ACCURACY",accuracy_score(y_test,test_predictions))
print("\nTest data f1-score for class '1'",f1_score(y_test,test_predictions,pos_label=
print("\nTest data f1-score for class '2'",f1_score(y_test,test_predictions,pos_label=
```

```
TRAIN Conf Matrix :
 [[13  7]
 [ 3 90]]

TRAIN DATA ACCURACY 0.911504424778761

Train data f1-score for class '1' 0.7222222222222222

Train data f1-score for class '2' 0.9473684210526315


---------------------------------------


TEST Conf Matrix :
 [[ 0  6]
 [ 0 23]]

TEST DATA ACCURACY 0.7931034482758621

Test data f1-score for class '1' 0.0

Test data f1-score for class '2' 0.8846153846153846
```

ANN

Used keras mnist dataset

In [70]:
```
import tensorflow
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense,Flatten
```

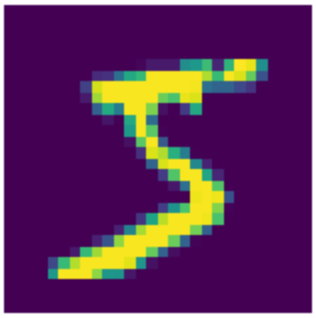
In [71]:
```
(X_train,y_train),(X_test,y_test)=keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

In [72]:
```
X_train.shape
```

Out[72]:
```
(60000, 28, 28)
```

In [74]:
```
plt.figure(figsize=(2, 2))
plt.imshow(X_train[0])
plt.axis('off')
plt.show()
```



In [75]:
```
X_train=X_train/255
X_test=X_test/255
```

In [76]:
```
#print(X_train[0])
```

In [77]:
```
model = Sequential()
model.add(Flatten(input_shape=(28,28)))
model.add(Dense(128,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(128,activation='softmax'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
```

In [78]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Pa
flatten (Flatten)	(None, 784)	
dense (Dense)	(None, 128)	1(
dense_1 (Dense)	(None, 32)	
dense_2 (Dense)	(None, 128)	

Total params: 108,832 (425.12 KB)

Trainable params: 108,832 (425.12 KB)

In [79]:
```
model.compile(loss='sparse_categorical_crossentropy',optimizer='Adam',metrics=['accura
```

In [81]:
```
history = model.fit(X_train, y_train, epochs=25, validation_split=0.2, verbose=0)

# Display the last 5 epochs' results
for epoch in range(-5, 0):
    print(f"Epoch {epoch + 25}: accuracy={history.history['accuracy'][epoch]:.4f}, "
          f"loss={history.history['loss'][epoch]:.4f}, "
          f"val_accuracy={history.history['val_accuracy'][epoch]:.4f}, "
          f"val_loss={history.history['val_loss'][epoch]:.4f}")
```

```
Epoch 20: accuracy=0.9976, loss=0.0082, val_accuracy=0.9743, val_loss=0.2065
Epoch 21: accuracy=0.9993, loss=0.0024, val_accuracy=0.9765, val_loss=0.2006
Epoch 22: accuracy=0.9980, loss=0.0071, val_accuracy=0.9734, val_loss=0.2172
Epoch 23: accuracy=0.9981, loss=0.0057, val_accuracy=0.9743, val_loss=0.2077
Epoch 24: accuracy=0.9983, loss=0.0055, val_accuracy=0.9750, val_loss=0.1973
```

In [82]:
```
y_prob = model.predict(X_test)
```

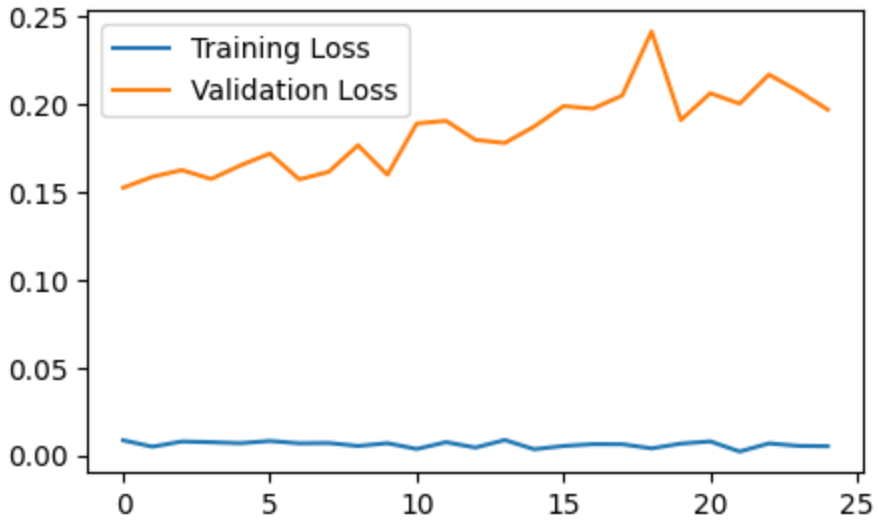
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```

In [83]:
```
y_pred = y_prob.argmax(axis=1)
```

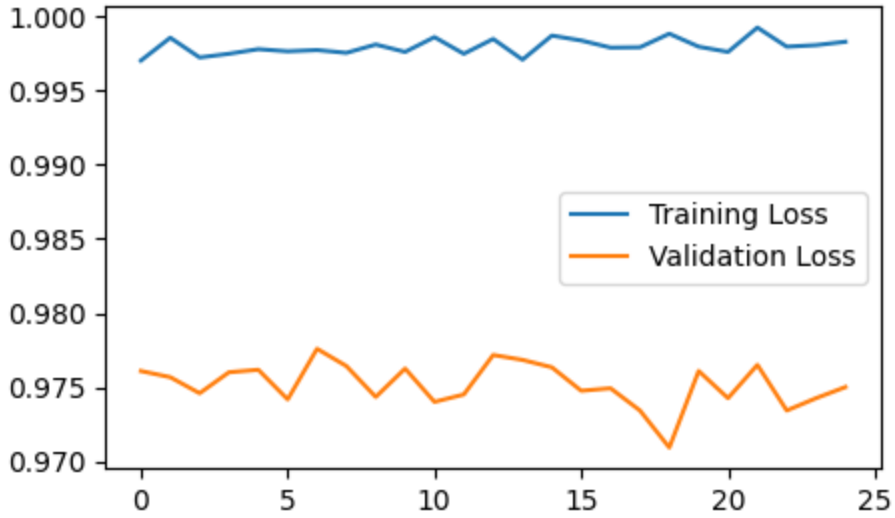
In [84]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

Out[84]:
```
0.9766
```

In [90]:
```
plt.figure(figsize=(5, 3))  # Set a smaller figure size
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.show()
```



In [89]:
```
plt.figure(figsize=(5, 3))  # Set a smaller figure size
plt.plot(history.history['accuracy'], label='Training Loss')
plt.plot(history.history['val_accuracy'], label='Validation Loss')
plt.legend()
plt.show()
```



In [92]:
```
model.predict(X_test[1].reshape(1,28,28)).argmax(axis=1)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
```

Out[92]:
```
array([2])
```

CNN

Used cat dog image dataset for classifying cat and dog

In [2]:
```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
```

```
cp: cannot stat 'kaggle.json': No such file or directory
```

In [3]:
```
!kaggle datasets download -d salader/dogs-vs-cats
```

```
Dataset URL: https://www.kaggle.com/datasets/salader/dogs-vs-cats
License(s): unknown
Downloading dogs-vs-cats.zip to /content
100% 1.06G/1.06G [00:17<00:00, 64.5MB/s]
100% 1.06G/1.06G [00:17<00:00, 66.6MB/s]
```

In [4]:
```
import zipfile
zip_ref=zipfile.ZipFile('/content/dogs-vs-cats.zip','r')
zip_ref.extractall('/content')
zip_ref.close()
```

In [5]:
```
import tensorflow as tf
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense,Flatten,Conv2D,MaxPooling2D, BatchNormalization, Dropou
```

In [6]:
```
# generators
train_ds=keras.utils.image_dataset_from_directory(
    directory='/content/train',
    labels="inferred",
    label_mode="int",
    batch_size=32,
    image_size=(256, 256)
)

validation_ds=keras.utils.image_dataset_from_directory(
    directory='/content/test',
    labels="inferred",
    label_mode="int",
    batch_size=32,
    image_size=(256, 256)
)
```

```
Found 20000 files belonging to 2 classes.
Found 5000 files belonging to 2 classes.
```

In [7]:
```
# normalized
def process(image,label):
  # cast(x,dtype)
  tf.cast(image/255.,tf.float32)
  return image,label

train_ds=train_ds.map(process)
validation_ds=validation_ds.map(process)
```

In [8]:
```
# create CNN model
model=Sequential()
model.add(Conv2D(32,kernel_size=(3,3),padding="valid",activation='relu',input_shape=(2
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding='valid'))

model.add(Conv2D(64,kernel_size=(3,3),padding="valid",activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding="valid"))

model.add(Conv2D(128,kernel_size=(3,3),padding="valid",activation='relu'))
model.add(BatchNormalization())
```

```
model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding="valid"))

model.add(Flatten())

model.add(Dense(128,activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(64,activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(1,activation='sigmoid'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:1
07: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When u
sing Sequential models, prefer using an `Input(shape)` object as the first layer in t
he model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [9]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Pa
conv2d (Conv2D)	(None, 254, 254, 32)	
batch_normalization (BatchNormalization)	(None, 254, 254, 32)	
max_pooling2d (MaxPooling2D)	(None, 127, 127, 32)	
conv2d_1 (Conv2D)	(None, 125, 125, 64)	1
batch_normalization_1 (BatchNormalization)	(None, 125, 125, 64)	
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 64)	
conv2d_2 (Conv2D)	(None, 60, 60, 128)	7
batch_normalization_2 (BatchNormalization)	(None, 60, 60, 128)	
max_pooling2d_2 (MaxPooling2D)	(None, 30, 30, 128)	
flatten (Flatten)	(None, 115200)	
dense (Dense)	(None, 128)	14,74
dropout (Dropout)	(None, 128)	
dense_1 (Dense)	(None, 64)	
dropout_1 (Dropout)	(None, 64)	
dense_2 (Dense)	(None, 1)	

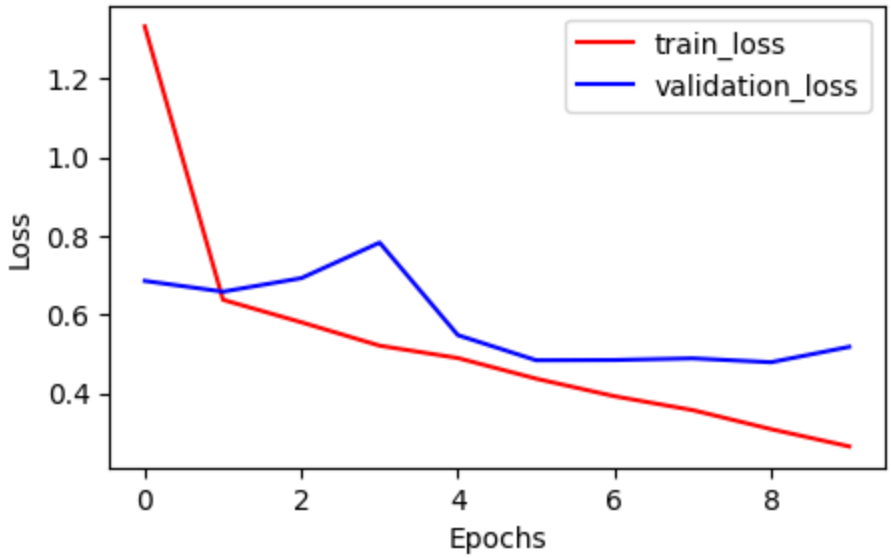
Total params: 14,848,193 (56.64 MB)

Trainable params: 14,847,745 (56.64 MB)

In [10]:
```
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
history=model.fit(train_ds,epochs=10,validation_data=validation_ds)
```

```
Epoch 1/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 65s 84ms/step - accuracy: 0.5539 - loss: 2.3986 - val_ac
curacy: 0.6114 - val_loss: 0.6851
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 50s 80ms/step - accuracy: 0.6287 - loss: 0.6544 - val_ac
curacy: 0.5918 - val_loss: 0.6578
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 81s 78ms/step - accuracy: 0.6998 - loss: 0.5878 - val_ac
curacy: 0.6598 - val_loss: 0.6921
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 83s 80ms/step - accuracy: 0.7489 - loss: 0.5276 - val_ac
curacy: 0.6592 - val_loss: 0.7825
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 81s 78ms/step - accuracy: 0.7690 - loss: 0.5085 - val_ac
curacy: 0.7354 - val_loss: 0.5471
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 83s 80ms/step - accuracy: 0.8033 - loss: 0.4436 - val_ac
curacy: 0.7830 - val_loss: 0.4832
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 85s 84ms/step - accuracy: 0.8323 - loss: 0.3984 - val_ac
curacy: 0.7908 - val_loss: 0.4838
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 82s 85ms/step - accuracy: 0.8528 - loss: 0.3647 - val_ac
curacy: 0.7794 - val_loss: 0.4881
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 82s 84ms/step - accuracy: 0.8781 - loss: 0.3148 - val_ac
curacy: 0.7992 - val_loss: 0.4783
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 82s 84ms/step - accuracy: 0.9006 - loss: 0.2702 - val_ac
curacy: 0.7950 - val_loss: 0.5175
```

In [15]:
```
plt.figure(figsize=(5, 3))  # Specify smaller width and height in inches
plt.plot(history.history['loss'], color='red', label='train_loss')
plt.plot(history.history['val_loss'], color='blue', label='validation_loss')
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```



In [16]:
```
model=Sequential()
model.add(Conv2D(32,kernel_size=(3,3),padding="valid",activation='relu',input_shape=(2
model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding='valid'))

model.add(Conv2D(64,kernel_size=(3,3),padding="valid",activation='relu',kernel_regular
model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding="valid"))

model.add(Conv2D(128,kernel_size=(3,3),padding="valid",activation='relu',kernel_regula
model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding="valid"))

model.add(Flatten())

model.add(Dense(128,activation='relu',kernel_regularizer=tf.keras.regularizers.l2(0.01
model.add(Dense(64,activation='relu',kernel_regularizer=tf.keras.regularizers.l2(0.01)

model.add(Dense(1,activation='sigmoid'))
```

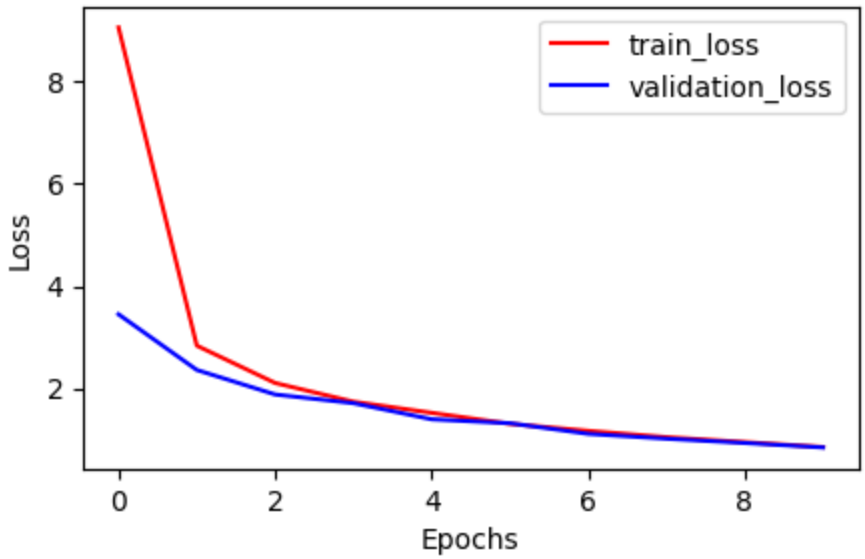
```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:1
07: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When u
sing Sequential models, prefer using an `Input(shape)` object as the first layer in t
he model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [17]:
```
model.compile(loss='binary_crossentropy',optimizer='Adam',metrics=['accuracy'])
history=model.fit(train_ds,epochs=10,validation_data=validation_ds)
```

```
Epoch 1/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 52s 76ms/step - accuracy: 0.5469 - loss: 30.1106 - val_a
ccuracy: 0.5786 - val_loss: 3.4499
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 78s 74ms/step - accuracy: 0.6065 - loss: 3.1066 - val_ac
curacy: 0.6422 - val_loss: 2.3612
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 81s 73ms/step - accuracy: 0.6469 - loss: 2.2354 - val_ac
curacy: 0.6552 - val_loss: 1.8818
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 82s 73ms/step - accuracy: 0.6677 - loss: 1.8072 - val_ac
curacy: 0.6216 - val_loss: 1.7119
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 82s 74ms/step - accuracy: 0.6823 - loss: 1.6040 - val_ac
curacy: 0.7328 - val_loss: 1.3967
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 89s 85ms/step - accuracy: 0.7377 - loss: 1.3463 - val_ac
curacy: 0.7160 - val_loss: 1.3196
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 80s 82ms/step - accuracy: 0.7590 - loss: 1.2161 - val_ac
curacy: 0.7730 - val_loss: 1.1123
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 79s 78ms/step - accuracy: 0.7804 - loss: 1.0757 - val_ac
curacy: 0.7772 - val_loss: 1.0171
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 78s 71ms/step - accuracy: 0.7923 - loss: 0.9739 - val_ac
curacy: 0.7792 - val_loss: 0.9337
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 83s 73ms/step - accuracy: 0.8051 - loss: 0.8757 - val_ac
curacy: 0.7824 - val_loss: 0.8463
```

In [19]:

```
plt.figure(figsize=(5, 3))  # Adjust to your preferred dimensions
plt.plot(history.history['loss'], color='red', label='train_loss')
plt.plot(history.history['val_loss'], color='blue', label='validation_loss')
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```



RNN

use seattle-weather.csv dataset from Kaggle

In [110…
```
df = pd.read_csv('seattle-weather.csv')
df.head()
```

Out[110]:

	date	precipitation	temp_max	temp_min	wind	weather
0	2012-01-01	0.0	12.8	5.0	4.7	drizzle
1	2012-01-02	10.9	10.6	2.8	4.5	rain
2	2012-01-03	0.8	11.7	7.2	2.3	rain
3	2012-01-04	20.3	12.2	5.6	4.7	rain
4	2012-01-05	1.3	8.9	2.8	6.1	rain

In [111…
```
df.isnull().sum()
```

Out[111]:

	0
date	0
precipitation	0
temp_max	0
temp_min	0
wind	0
weather	0

dtype: int64

In [112…
```
df.duplicated().sum()
```

Out[112]:
```
0
```

In [113…
```
training_set = df.iloc[:,2:3].values
training_set
```

Out[113]:
```
array([[12.8],
       [10.6],
       [11.7],
       ...,
       [ 7.2],
       [ 5.6],
       [ 5.6]])
```

In [114…
```
len(training_set)
```

Out[114]:
```
1461
```

In [115…
```
def df_to_XY(df,window_size=10):
 X_train=[]
 y_train=[]

 for i in range(10,len(training_set)):
    X_train.append(training_set[i-10:i,0])
    y_train.append(training_set[i,0])

 X_train, y_train = np.array(X_train), np.array(y_train)
 return X_train, y_train
```

In [116…
```
WINDOW = 10
X,y = df_to_XY(df,WINDOW)
print(len(X),len(y))
X_train = X[:800]
y_train = y[:800]
X_val = X[800:1000]
y_val = y[800:1000]
X_test = X[1000:]
x_test = y[1000:]
```

```
1451 1451
```

In [117…
```
#Reshaping(To add new dimensions)
X_train = np.reshape(X_train,(X_train.shape[0],X_train.shape[1],1))
X_val = np.reshape(X_val,(X_val.shape[0],X_val.shape[1],1))
X_test = np.reshape(X_test,(X_test.shape[0],X_test.shape[1],1))
```

In [118…
```
#Building the RNN
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout
```

In [119…
```
regressor = Sequential()
```

In [120…
```
#Addinf the first LSTM layer and some Dropout regularisation
regressor.add(LSTM(units=50, return_sequences = True, input_shape=(X_train.shape[1], 1
regressor.add(Dropout(0.2))

regressor.add(LSTM(units=50, return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units=50, return_sequences = True))
regressor.add(Dropout(0.2))
regressor.add(LSTM(units=50))
regressor.add(Dropout(0.2))
#Output layer
regressor.add(Dense(units=1))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential m
odels, prefer using an `Input(shape)` object as the first layer in the model instead.
```

In [121…
```
#Compiling
regressor.compile(optimizer='adam',loss='mean_squared_error')
```

In [130…
```
from tensorflow.keras.callbacks import ModelCheckpoint,EarlyStopping
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.metrics import RootMeanSquaredError
from tensorflow.keras.optimizers import Adam

# Fit the model with verbose=0 to suppress output
history = regressor.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=100,

# Display results for the last 5 epochs
last_5_epochs_loss = history.history['loss'][-5:]
last_5_epochs_val_loss = history.history['val_loss'][-5:]

print("Last 5 Epochs' Training and Validation Loss:")
for i, (loss, val_loss) in enumerate(zip(last_5_epochs_loss, last_5_epochs_val_loss),
    epoch_num = 100 - 5 + i  # Calculate epoch number for display
    print(f"Epoch {epoch_num:3d}: Training Loss = {loss:.4f}, Validation Loss = {val_l
```

```
Last 5 Epochs' Training and Validation Loss:
Epoch  96: Training Loss = 2.7545, Validation Loss = 13.2436
Epoch  97: Training Loss = 2.4732, Validation Loss = 13.8759
Epoch  98: Training Loss = 2.6289, Validation Loss = 13.3852
Epoch  99: Training Loss = 2.5229, Validation Loss = 13.9444
Epoch 100: Training Loss = 2.8279, Validation Loss = 13.8125
```

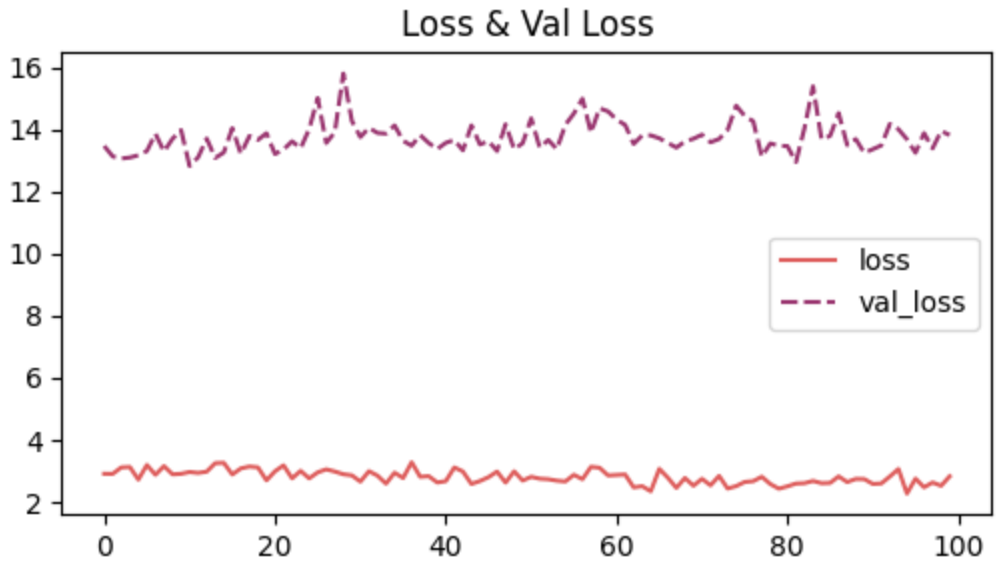
In [131…
```
his = pd.DataFrame(history.history)
```

In [132…
```
his.head()
```

Out[132]:

	loss	val_loss
0	2.909266	13.461831
1	2.906893	13.124987
2	3.113686	13.059484
3	3.126990	13.089668
4	2.722066	13.163455

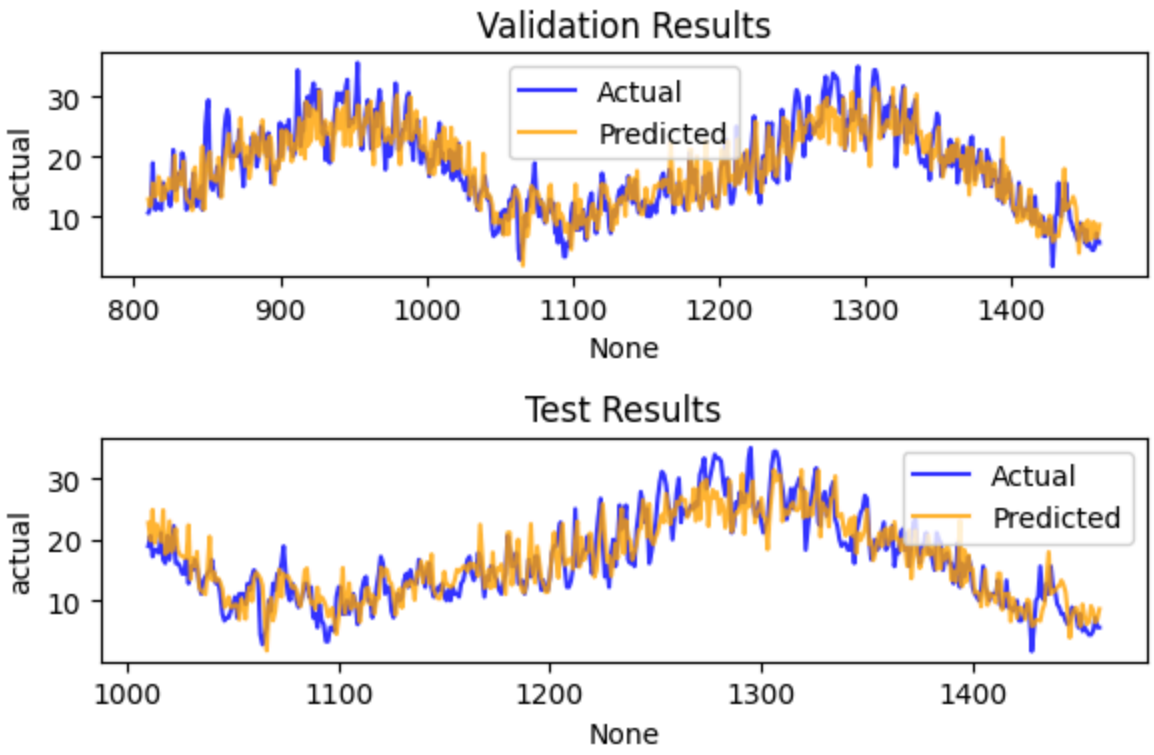
In [138…
```
import seaborn as sns
his.columns
history_loss = his[['loss', 'val_loss']]
fig,axes = plt.subplots(1,1,figsize=(6,3))
plt.subplot(1,1,1)
plt.title("Loss & Val Loss")
sns.lineplot(history_loss,palette="flare");
```



In [139…
```
train_pred = regressor.predict(X_train).flatten()
val_pred = regressor.predict(X_val).flatten()
test_pred = regressor.predict(X_test).flatten()
```

```
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 109ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
```

In [143…
```
fig, axes = plt.subplots(2, 1, figsize=(6, 4), dpi=100)

# Validation Results
plt.subplot(2, 1, 1)
plt.title("Validation Results")
sns.lineplot(data=df_pred[800:], x=df_pred.index[800:], y="actual", color="blue", labe
sns.lineplot(data=df_pred[800:], x=df_pred.index[800:], y="predicted", color="orange",

# Test Results
plt.subplot(2, 1, 2)
plt.title("Test Results")
sns.lineplot(data=df_pred[1000:], x=df_pred.index[1000:], y="actual", color="blue", la
sns.lineplot(data=df_pred[1000:], x=df_pred.index[1000:], y="predicted", color="orange

plt.tight_layout()  # Adjust layout for a better fit
plt.legend()        # Show legend for labels
plt.show()
```



LSTM(Long Short Term Memory)

used two dataset from Kaggle 'DailyDelhiClimateTrain.csv' and 'DailyDelhiClimateTest.csv

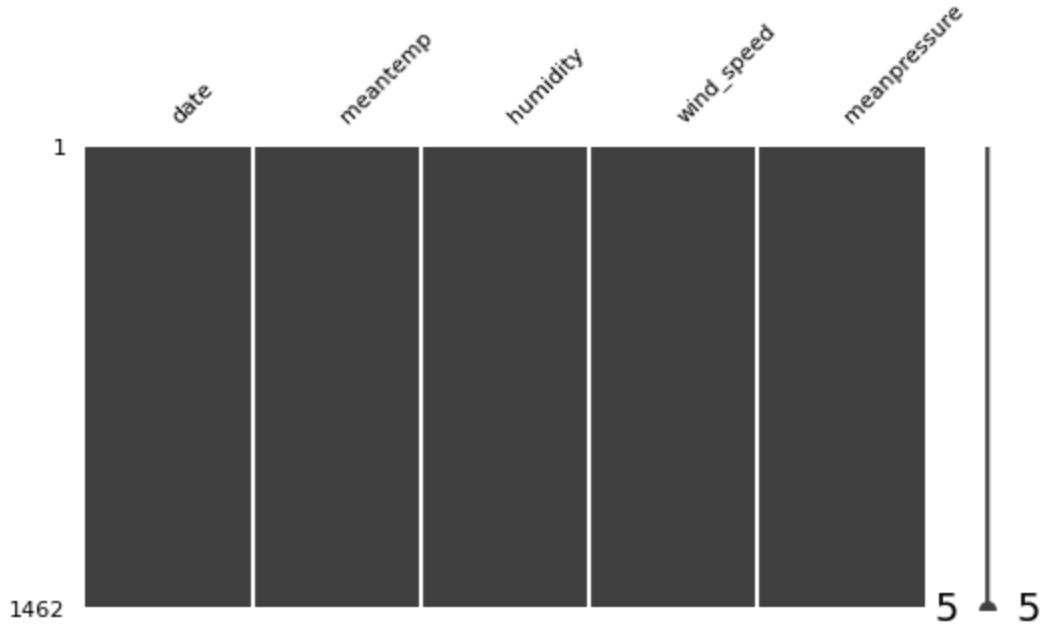
In [120…
```
train = pd.read_csv('DailyDelhiClimateTrain.csv')
test = pd.read_csv('DailyDelhiClimateTest.csv')
train.head()
```

Out[120]:

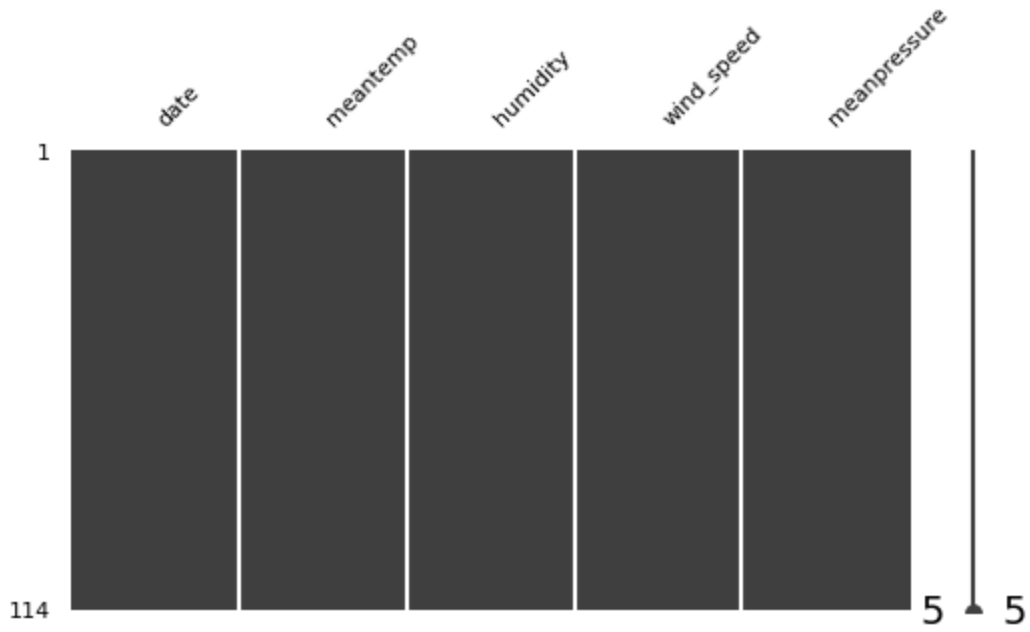
	date	meantemp	humidity	wind_speed	meanpressure
0	2013-01-01	10.000000	84.500000	0.000000	1015.666667
1	2013-01-02	7.400000	92.000000	2.980000	1017.800000
2	2013-01-03	7.166667	87.000000	4.633333	1018.666667
3	2013-01-04	8.666667	71.333333	1.233333	1017.166667
4	2013-01-05	6.000000	86.833333	3.700000	1016.500000

In [121…
```
import missingno as m
plt.figure(figsize=(6, 3))
m.matrix(train, figsize=(6, 3))
plt.tick_params(axis='both', which='major', labelsize=8)
plt.show()

plt.figure(figsize=(6, 3))
m.matrix(test, figsize=(6, 3))
plt.tick_params(axis='both', which='major', labelsize=8)
plt.show()
```

```
<Figure size 600x300 with 0 Axes>
```




```
<Figure size 600x300 with 0 Axes>
```



In [122…

```
train.describe()
test.describe()
```

Out[122]:

	meantemp	humidity	wind_speed	meanpressure
count	114.000000	114.000000	114.000000	114.000000
mean	21.713079	56.258362	8.143924	1004.035090
std	6.360072	19.068083	3.588049	89.474692
min	11.000000	17.750000	1.387500	59.000000
25%	16.437198	39.625000	5.563542	1007.437500
50%	19.875000	57.750000	8.069444	1012.739316
75%	27.705357	71.902778	10.068750	1016.739583
max	34.500000	95.833333	19.314286	1022.809524

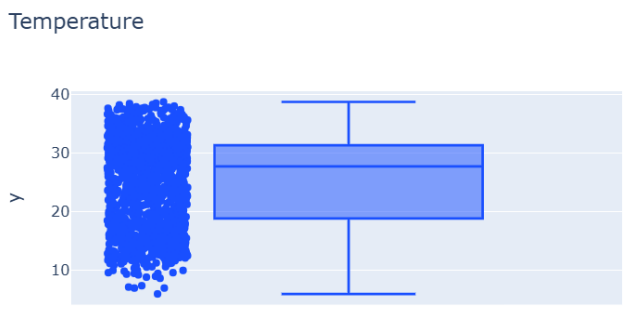
In [123…

```
print(train.shape)
print(test.shape)
```

```
(1462, 5)
(114, 5)
```

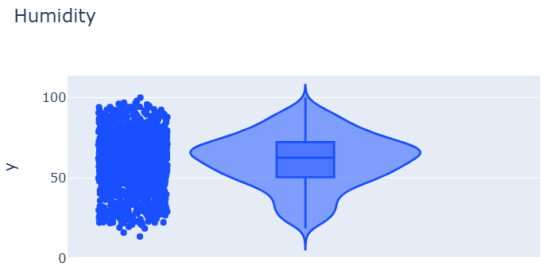
In [134…

```
#Explanatory Data Analysis (EDA)
import plotly.express as px
fig = px.box(y=train['meantemp'],points='all',title='Temperature',width=600,height=350
fig.show()
```

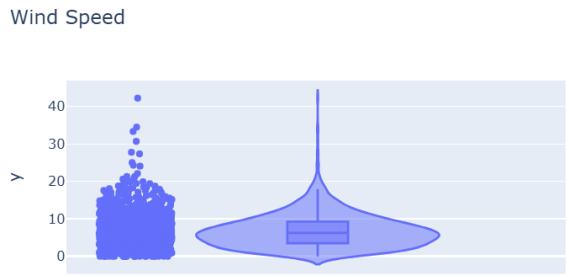


In [97]:

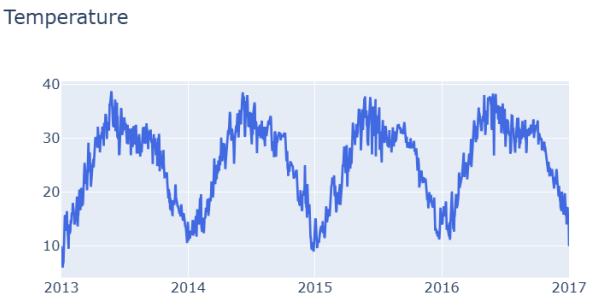
```
fig = px.violin(y=train['humidity'],points='all',box=True,title='Humidity',
width=600,height=350,
    color_discrete_sequence=['#194fff']
)
fig.show()
```



In [98]:
```
fig = px.violin(y=train['wind_speed'], points='all',box=True, title='Wind Speed', widt
fig.show()
```

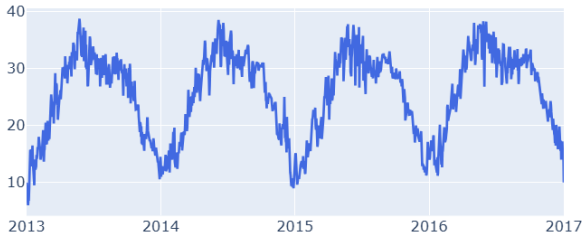


In [99]:
```
#Temperature
import plotly.graph_objects as go
fig = go.Figure()
fig.add_trace(go.Scatter(x=train['date'], y=train['meantemp'], name='Real Price',
                         line=dict(color='royalblue', width=2)))

fig.update_layout(title='Temperature', width=600, height=350)  # Adjust width and heig
fig.show()
```



In [100…

```
import plotly.graph_objects as go
fig2 = go.Figure()

fig2.add_trace(go.Scatter( x=train['date'], y=train['humidity'], name = 'Predicted Pri
                          line=dict(color='firebrick', width=2)))
fig.update_layout(title='Humidity', width=600, height=350)
fig.show()
```



In [79]:

```
temp_train = train.iloc[:,1:2]
temp_test = test.iloc[:,1:2]
```

In [80]:
```
from sklearn.preprocessing import MinMaxScaler
ss= MinMaxScaler(feature_range=(0,1))
temp_train= ss.fit_transform(temp_train)
temp_test= ss.fit_transform(temp_test)
```

In [81]:
```
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return np.array(dataX), np.array(dataY)
```

In [82]:
```
look_back = 1
trainX, trainY = create_dataset(temp_train, look_back)
testX, testY = create_dataset(temp_test, look_back)
```

In [83]:
```
trainX = np.reshape(trainX, (trainX.shape[0], trainX.shape[1],1))
testX = np.reshape(testX, (testX.shape[0], testX.shape[1],1))
```

Building the LSTM model

In [84]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, LSTM, Dropout, Dense

# Initialize the model
model_temp = Sequential()
model_temp.add(Input(shape=(trainX.shape[1], 1)))
model_temp.add(LSTM(units=100, return_sequences=True))
model_temp.add(Dropout(0.2))

model_temp.add(LSTM(units=100, return_sequences=True))
model_temp.add(Dropout(0.2))

model_temp.add(LSTM(units=100, return_sequences=True))
model_temp.add(Dropout(0.2))

model_temp.add(LSTM(units=50))
model_temp.add(Dropout(0.2))
model_temp.add(Dense(units=1))
model_temp.compile(optimizer='adam', loss='mean_squared_error')
```

In [85]:
```
#Compiling with ADAM Optimizer and Mean Squared Error
# Compiling the RNN
model_temp.compile(optimizer = 'adam', loss = 'mean_squared_error', metrics=['accuracy
# Fitting the RNN to the Training set
history = model_temp.fit(trainX, trainY, epochs = 100, batch_size = 32,verbose=0)
for i in range(95, 100):
    print(f"Epoch {i+1}/100 - loss: {history.history['loss'][i]:.4f} - accuracy: {hist
```

```
Epoch 96/100 - loss: 0.0034 - accuracy: 0.0014
Epoch 97/100 - loss: 0.0036 - accuracy: 0.0014
Epoch 98/100 - loss: 0.0034 - accuracy: 0.0014
Epoch 99/100 - loss: 0.0035 - accuracy: 0.0014
Epoch 100/100 - loss: 0.0036 - accuracy: 0.0014
```

In [86]:
```
#Prediction for Test Set
prediction = model_temp.predict(testX)
prediction = ss.inverse_transform(prediction)
temp_test = ss.inverse_transform(temp_test)
#prediction
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 193ms/step
```

In [87]:
```
temp_test.shape
```

Out[87]:
```
(114, 1)
```

In [88]:
```
prediction.shape
```

Out[88]:
```
(112, 1)
```

In [89]:
```
prediction1 = prediction
```

In [90]:
```
prediction.mean()
```

Out[90]:
```
21.748611
```

In [91]:
```
prediction1 = np.append(prediction1, [21.90234,21.90234])
```

In [92]:
```
prediction1

prediction1 = prediction1.reshape(len(prediction1),1)
prediction1.shape
```

Out[92]:
```
(114, 1)
```

In [93]:
```
data = np.concatenate((prediction1.reshape(len(prediction1),1), temp_test.reshape(len(
#print(data)
```

In [94]:
```
from sklearn.metrics import r2_score
meantemp = r2_score(temp_test, prediction1)
print("Accuracy of Temperature : ", meantemp)
```

```
Accuracy of Temperature :  0.9395911399801052
```