APPLICATION OF MACHINE LEARNING IN INDUSTRIES LAB

Name: Nidhi Mishra

SAP ID: 500097158

Roll no: R2142211276

Batch: B6

SUBMISSION TO:

Dr. Pallabi Sharma

Assistant Professor

School of Computer Science

UPES, Dehradun

S.No.	**Title**	**Page No.**
1	Introduction to Pandas and Numpy.	3-10
2	Wine quality prediction.	11-14
3	House Price prediction.	15-16
4	Air Quality Prediction.	17-20
5	Credit Card fault prediction.	21-23
6	Hypothesis Testing.	24-26
7	Neural Network (LSTM) model on sequential dataset.	27-28
8	Compare the LSTM (Neural network model) &CNN (Deep learning model).	29-31

EXPERIMENT-1
Introduction to Pandas and Numpy

CODE-

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

 # Task 1: Basic DataFrame Operations

```
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data', header=None, names=['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class'])


print(df.head())
print("\nMissing values:")
print(df.isnull().sum())


df.dropna(inplace=True)


print("\nSummary of the dataset:")
print(df.describe())


subset_label = df[['sepal_length', 'sepal_width', 'class']]
subset_position = df.iloc[:, [0, 1, 4]]
conditioned_df = df[df['sepal_length'] > 5.0]
```

 # Task 2: Data Cleaning and Preprocessing

```
missing_values = df.isnull().sum()
df.fillna(df.mean(), inplace=True

df['class_encoded'] = df['class'].astype('category').cat.codes

grouped_data = df.groupby('class')
aggregated_results = grouped_data.agg({'sepal_length': ['sum',
'ean', 'count']})
```

 # Task 3: Merge two different datasets using different typesof joins

```
df1 = pd.read_csv('dataset1.csv')
df2 = pd.read_csv('dataset2.csv')

inner_join = pd.merge(df1, df2, how='inner', on='common_column')
outer_join = pd.merge(df1, df2, how='outer', on='common_column')
left_join = pd.merge(df1, df2, how='left', on='common_column')
right_join = pd.merge(df1, df2, how='right', on='common_column')
```

 # Task 4: Visualization

```
df.plot.bar(x='class', y='sepal_length', title='Bar Plot')
df.plot.line(x='sepal_length', y='petal_length', title='Line Plot')
df.plot.scatter(x='sepal_length', y='petal_length', title='Scatter Plot')

correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')

df.hist()
df.boxplot()
```

 # Task 5: Basic NumPy Operations

```
arr = np.arange(1, 11)
arr2 = np.arange(11, 21)
```

```
add_result = arr + arr2
subtract_result = arr - arr2
```

```
multiply_result = arr * arr2
divide_result = arr / arr2
```

 # Task 6: Array Manipulation

```
arr_reshaped = arr.reshape(2, 5)
arr_transposed = arr_reshaped.T
arr_flattened = arr_transposed.flatten()
stacked_arrays = np.vstack((arr, arr2))
```

 # Task 7: Statistical Operations

```
mean_value = np.mean(arr)
median_value = np.median(arr)
std_deviation = np.std(arr)
max_value = np.max(arr)
min_value = np.min(arr)
normalized_arr = (arr - mean_value) / std_deviation
```

 # Task 8: Boolean Indexing

```
bool_arr = arr > 5
filtered_arr = arr[bool_arr]
```

 # Task 9: Random Module

```
random_matrix = np.random.rand(3, 3)
random_integers = np.random.randint(1, 100, 10)
np.random.shuffle(arr)
```

 # Task 10: Universal Functions (ufunc)

```
sqrt_arr = np.sqrt(arr)
exp_arr = np.exp(arr)
```

 # Task 11: Linear Algebra Operations

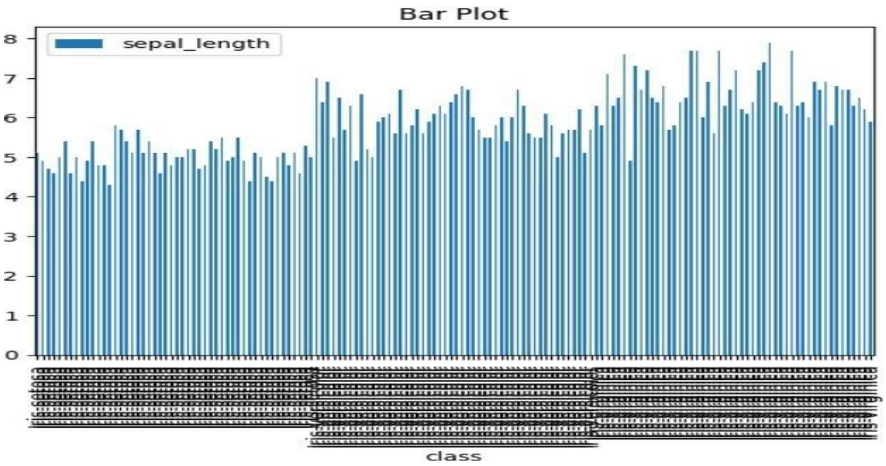
```
mat_a = np.random.rand(3, 3)
vec_b = np.random.rand(3, 1)
result = np.dot(mat_a, vec_b)
```

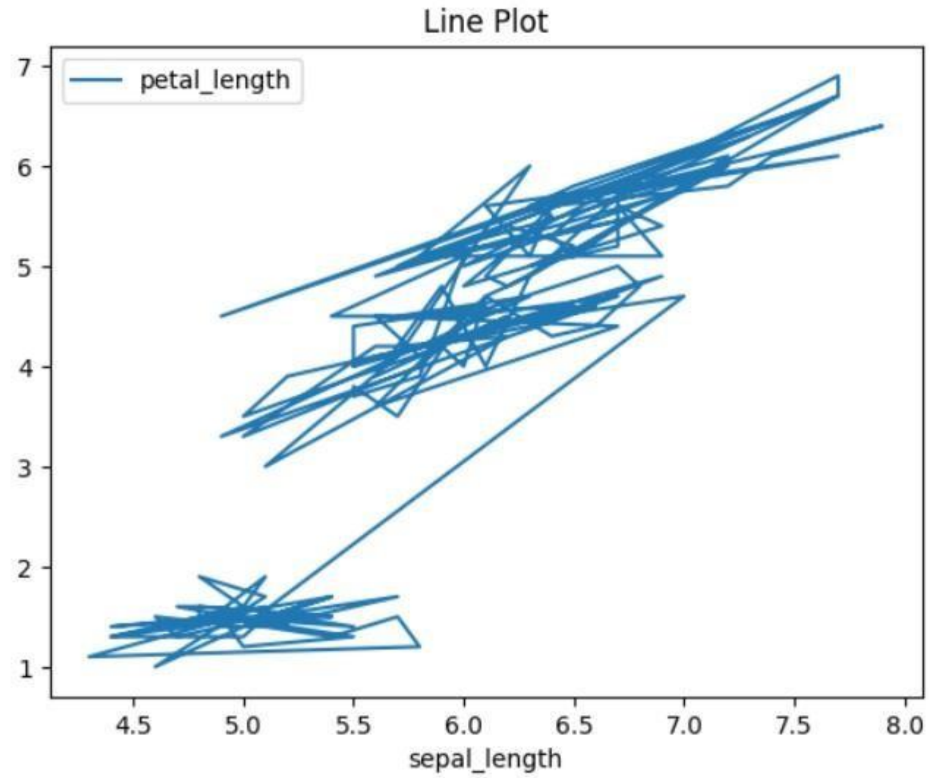
 # Task 12: Broadcasting

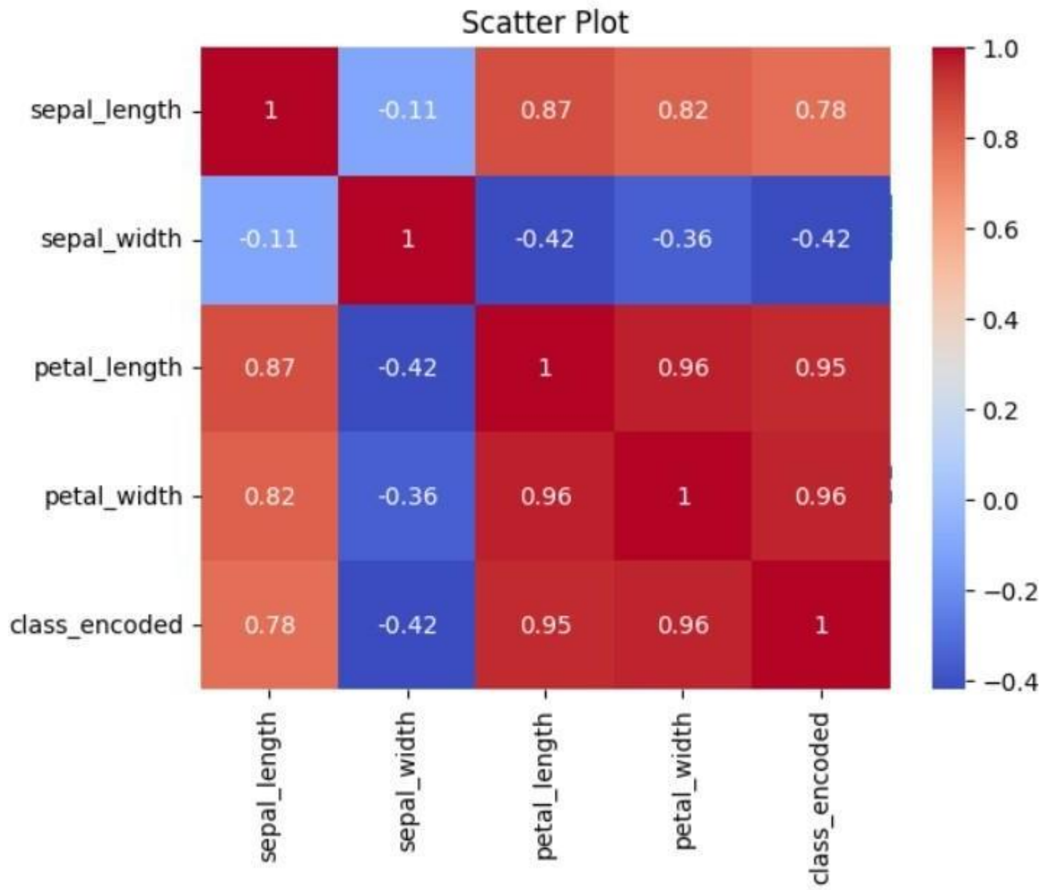
```
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
mean_per_row = matrix.mean(axis=1, keepdims=True)
result_broadcasting = matrix - mean_per_row

plt.show()
```

OUTPUT-

```
   sepal_length  sepal_width  petal_length  petal_width        class
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa

Missing values:
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
class           0
dtype: int64

Summary of the dataset:
       sepal_length  sepal_width  petal_length  petal_width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
```




Line Plot
petal_length
7
6
5
4
3
2
1
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
sepal_length


Scatter Plot
sepal_length
sepal_width
petal_length
petal_width
class_encoded
1 -0.11 0.87 0.82 0.78
-0.11 1 -0.42 -0.36 -0.42
0.87 -0.42 1 0.96 0.95
0.82 -0.36 0.96 1 0.96
0.78 -0.42 0.95 0.96 1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

EXPERIMENT-2
Wine quality prediction

CODE-

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns


wine_data = pd.read_csv("C:/Users/pc/Downloads/archive
(6)/WineQT.csv")

# Split features and target variable
X = wine_data.drop('quality', axis=1)
y = wine_data['quality']

# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Define hyperparameters for tuning
param_grid = {
    'n_estimators': [50, 100, 150],
```

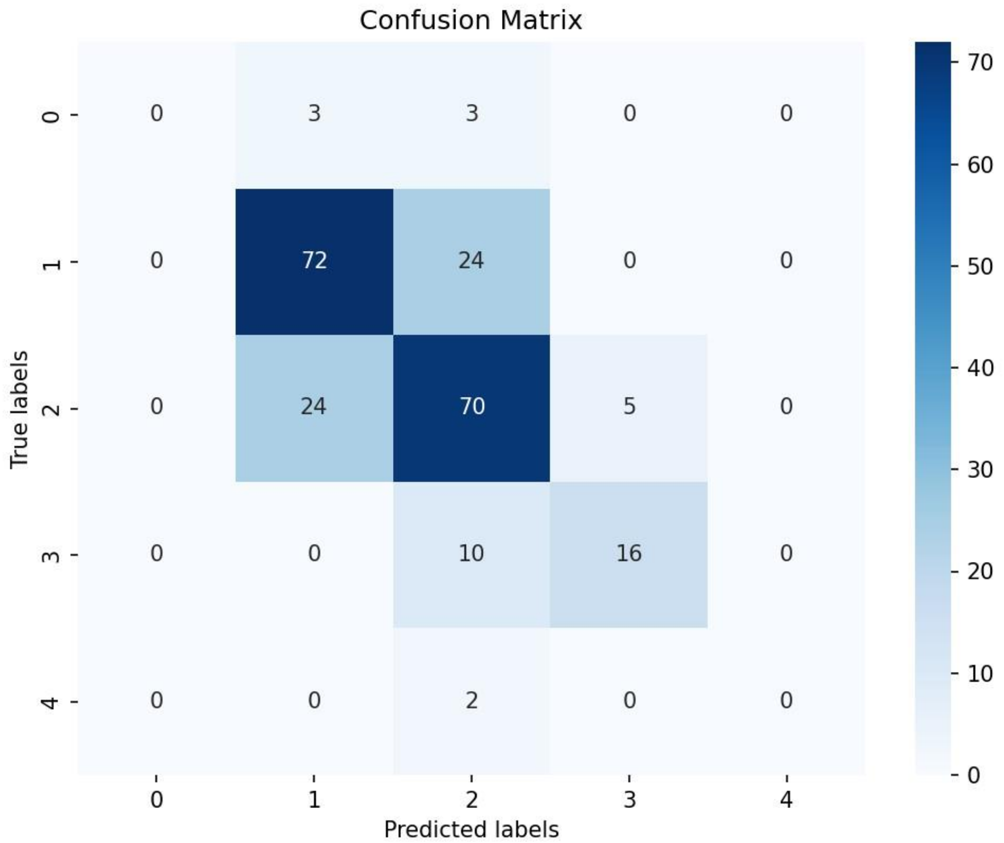
```
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Hyperparameter tuning using GridSearchCV
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=5)
grid_search.fit(X_train_scaled, y_train)

# Best parameters found
best_params = grid_search.best_params_
print("Best Parameters:", best_params)

# Train the classifier with best parameters
best_rf_classifier = RandomForestClassifier(**best_params, random_state=42)
best_rf_classifier.fit(X_train_scaled, y_train)

# Predict the test set results
y_pred = best_rf_classifier.predict(X_test_scaled)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Plotting feature importances
feature_importances = best_rf_classifier.feature_importances_
feature_names = X.columns
sorted_idx = feature_importances.argsort()

plt.figure(figsize=(10, 6))
```

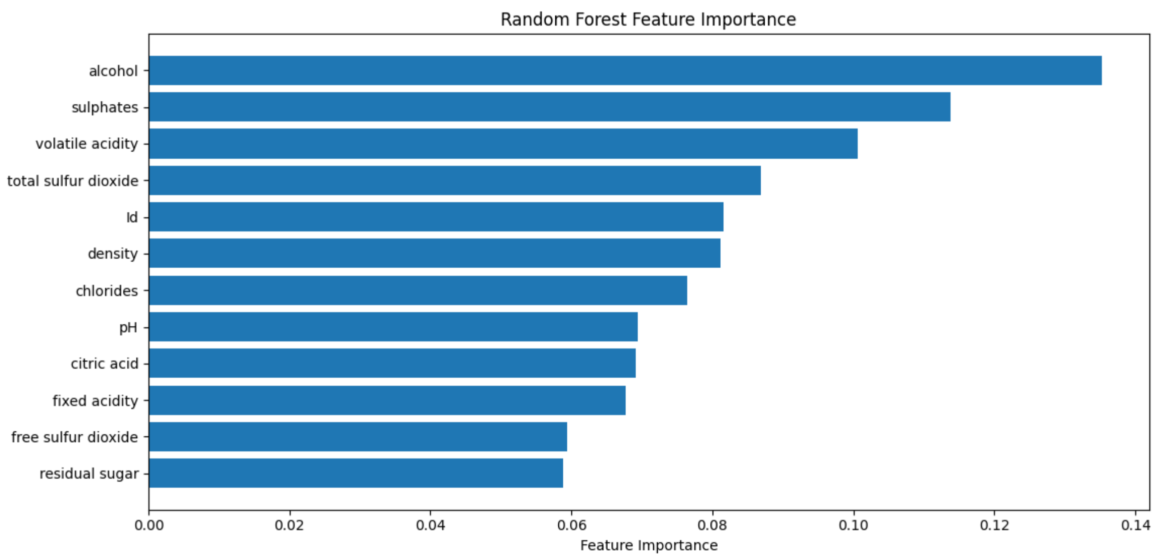
```
plt.barh(range(len(sorted_idx)), feature_importances[sorted_idx], align='center')
plt.yticks(range(len(sorted_idx)), [feature_names[i] for i in sorted_idx])
plt.xlabel('Feature Importance')
plt.title('Random Forest Feature Importance')
plt.show()

# Plotting confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

OUTPUT-

```
              precision    recall  f1-score   support

           4       0.00      0.00      0.00         6
           5       0.73      0.75      0.74        96
           6       0.64      0.71      0.67        99
           7       0.76      0.62      0.68        26
           8       0.00      0.00      0.00         2

    accuracy                           0.69       229
   macro avg       0.43      0.41      0.42       229
weighted avg       0.67      0.69      0.68       229
```

```
Best Parameters: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Accuracy: 0.6899563318777293
```


Confusion Matrix
True labels
Predicted labels
0 3 3 0 0
0 72 24 0 0
0 24 70 5 0
0 0 10 16 0
0 0 2 0 0
0
1
2
3
4
70
60
50
40
30
20
10


Random Forest Feature Importance
alcohol
sulphates
volatile acidity
total sulfur dioxide
Id
density
chlorides
pH
citric acid
fixed acidity
free sulfur dioxide
residual sugar
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
Feature Importance

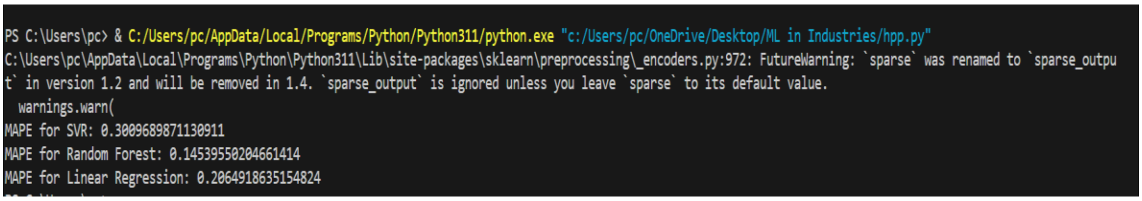
EXPERIMENT-3
House Price prediction

CODE-

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_percentage_error

# Read the dataset
dataset =
pd.read_excel("C:/Users/pc/Downloads/HousePricePrediction.xlsx")

# Drop 'Id' column and handle missing values
dataset.drop(['Id'], axis=1, inplace=True)
new_dataset = dataset.dropna()

# Identify categorical, integer, and float variables
object_cols =
new_dataset.select_dtypes(include=['object']).columns.tolist()

# Perform one-hot encoding for categorical variables
OH_encoder = OneHotEncoder(sparse=False)
OH_cols =
pd.DataFrame(OH_encoder.fit_transform(new_dataset[object_cols]))
OH_cols.index = new_dataset.index
OH_cols.columns = OH_encoder.get_feature_names_out(object_cols)
df_final = new_dataset.drop(object_cols, axis=1)
df_final = pd.concat([df_final, OH_cols], axis=1)

# Split dataset into features and target variable
X = df_final.drop(['SalePrice'], axis=1)
Y = df_final['SalePrice']
```

```
# Split the dataset into training and validation sets
X_train, X_valid, Y_train, Y_valid = train_test_split(X, Y, train_size=0.8, test_size=0.2, random_state=0)

# Train Support Vector Regression model
model_SVR = SVR()
model_SVR.fit(X_train, Y_train)
Y_pred_SVR = model_SVR.predict(X_valid)
print("MAPE for SVR:", mean_absolute_percentage_error(Y_valid, Y_pred_SVR))

# Train Random Forest Regression model
model_RFR = RandomForestRegressor(n_estimators=10)
model_RFR.fit(X_train, Y_train)
Y_pred_RFR = model_RFR.predict(X_valid)
print("MAPE for Random Forest:", mean_absolute_percentage_error(Y_valid, Y_pred_RFR))

# Train Linear Regression model
model_LR = LinearRegression()
model_LR.fit(X_train, Y_train)
Y_pred_LR = model_LR.predict(X_valid)
print("MAPE for Linear Regression:", mean_absolute_percentage_error(Y_valid, Y_pred_LR))
```

OUTPUT-



```
PS C:\Users\pc> & C:/Users/pc/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pc/OneDrive/Desktop/ML in Industries/hpp.py"
C:\Users\pc\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\preprocessing\_encoders.py:972: FutureWarning: `sparse` was renamed to `sparse_outpu
t` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
MAPE for SVR: 0.3009689871130911
MAPE for Random Forest: 0.14539550204661414
MAPE for Linear Regression: 0.2064918635154824
```

EXPERIMENT-4
Air Quality Prediction

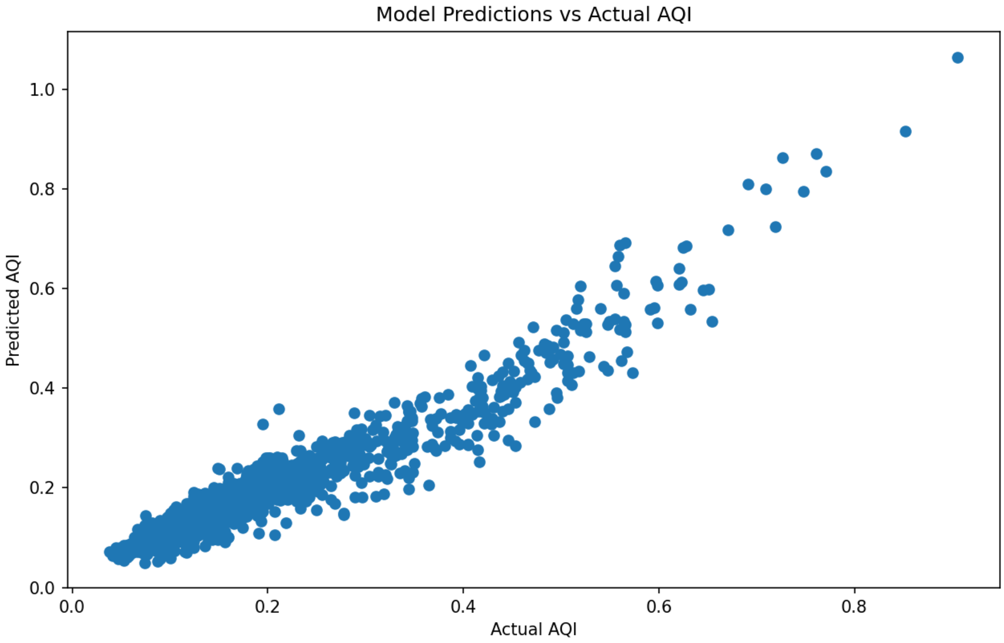
CODE-

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Load the dataset
city_day = pd.read_csv("C:/Users/pc/Downloads/archive (8)/city_day.csv")

# Drop rows with missing values
cleaned_data = city_day.dropna()

# Determine the number of unique cities in the dataset
no_of_cities = len(cleaned_data.City.unique())
print("Number of unique cities:", no_of_cities)

# Determine the unique AQI buckets in the dataset
aqi_buckets = cleaned_data.AQI_Bucket.unique()
print("Unique AQI Buckets:", aqi_buckets)

# Filter data for each AQI bucket
moderate = cleaned_data.AQI[cleaned_data.AQI_Bucket == 'Moderate']
poor = cleaned_data.AQI[cleaned_data.AQI_Bucket == 'Poor']
very_poor = cleaned_data.AQI[cleaned_data.AQI_Bucket == 'Very Poor']
satisfactory = cleaned_data.AQI[cleaned_data.AQI_Bucket == 'Satisfactory']
good = cleaned_data.AQI[cleaned_data.AQI_Bucket == 'Good']
severe = cleaned_data.AQI[cleaned_data.AQI_Bucket == 'Severe']
```

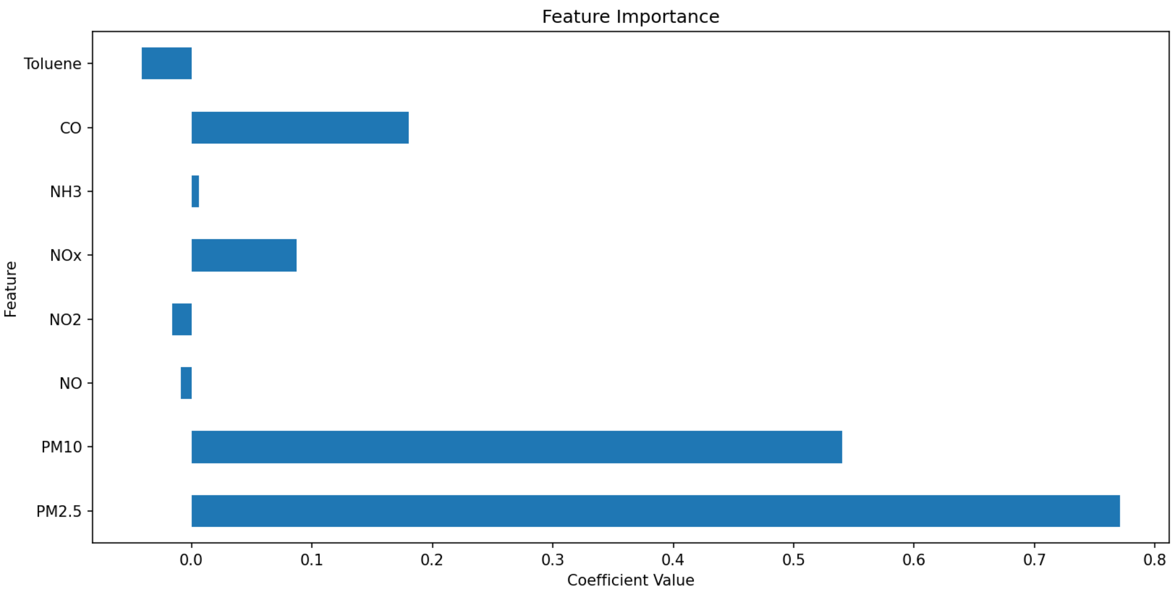
```
# Selecting numerical columns for correlation analysis
gasses = cleaned_data.select_dtypes(include=np.float64)

# Calculating correlation between gases and AQI
corr = gasses.corr().AQI

# Identifying columns to drop based on correlation threshold
col_to_drop = corr[abs(corr) < 0.45].index

# Droping columns with low correlation
gasses = gasses.drop(columns=col_to_drop)

# Defining target variable (AQI) and features (gases)
y = gasses.AQI
X = gasses.drop(columns='AQI')

# Normalize target variable (AQI)
y_max = y.max()
y = y / y_max

# Normalize features
X_max = X.max()
X = X / X_max

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Display model's performance
```

```
print("\nModel Evaluation:")
print("Mean Squared Error:", mse)
print("R-squared:", r2)

# Evaluate the model using cross-validation
cv_scores = cross_val_score(model, X_train, y_train, cv=5)
print("Cross-Validation Scores:", cv_scores)
print("Mean CV Score:", np.mean(cv_scores))

# Visualize model predictions vs actual air quality measurements
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred)
plt.xlabel("Actual AQI")
plt.ylabel("Predicted AQI")
plt.title("Model Predictions vs Actual AQI")
plt.show()

# Interpret trained model to understand feature importance
coefficients = pd.Series(model.coef_, index=X.columns)
print("Feature Importance:\n", coefficients)

# Visualize feature importances using bar plot
coefficients.plot(kind='barh')
plt.xlabel("Coefficient Value")
plt.ylabel("Feature")
plt.title("Feature Importance")
plt.show()
```

OUTPUT-



Model Predictions vs Actual AQI
Predicted AQI
Actual AQI
0.0
0.2
0.4
0.6
0.8
1.0



Feature Importance
Feature
Toluene
CO
NH3
NOx
NO2
NO
PM10
PM2.5
Coefficient Value
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

EXPERIMENT-5
Credit Card fault prediction

CODE-

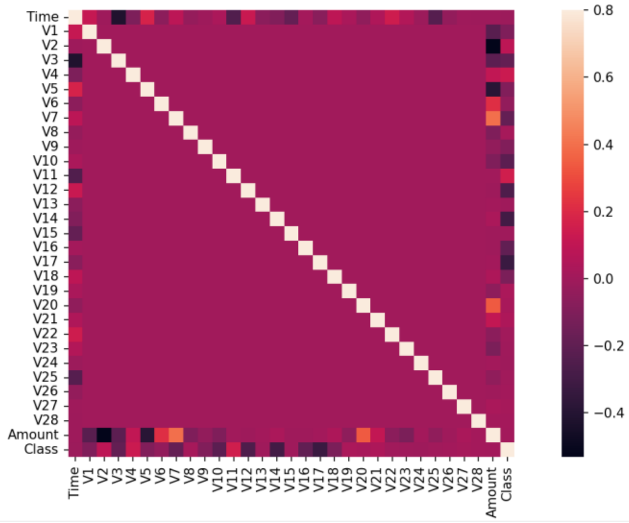
```
 import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import gridspec

data = pd.read_csv("C:/Users/pc/Downloads/archive
(12)/creditcard.csv")

data.head()

print(data.shape)
print(data.describe())

fraud = data[data['Class'] == 1]
valid = data[data['Class'] == 0]
outlierFraction = len(fraud)/float(len(valid))
print(outlierFraction)
print('Fraud Cases: {}'.format(len(data[data['Class'] == 1])))
print('Valid Transactions: {}'.format(len(data[data['Class'] == 0])))
print("Amount details of the fraudulent transaction")
fraud.Amount.describe()
print("details of valid transaction")
valid.Amount.describe()

corrmat = data.corr()
fig = plt.figure(figsize = (12, 9))
sns.heatmap(corrmat, vmax = .8, square = True)
plt.show()

X = data.drop(['Class'], axis = 1)
Y = data["Class"]
```

```
print(X.shape)
print(Y.shape)

xData = X.values
yData = Y.values

from sklearn.model_selection import train_test_split

xTrain, xTest, yTrain, yTest = train_test_split(
                xData, yData, test_size = 0.2, random_state = 42)

from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier()
rfc.fit(xTrain, yTrain)

yPred = rfc.predict(xTest)

from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import f1_score, matthews_corrcoef
from sklearn.metrics import confusion_matrix

n_outliers = len(fraud)
n_errors = (yPred != yTest).sum()
print("The model used is Random Forest classifier")

acc = accuracy_score(yTest, yPred)
print("The accuracy is {}".format(acc))

prec = precision_score(yTest, yPred)
print("The precision is {}".format(prec))

rec = recall_score(yTest, yPred)
print("The recall is {}".format(rec))

f1 = f1_score(yTest, yPred)
```

```
print("The F1-Score is {}".format(f1))

MCC = matthews_corrcoef(yTest, yPred)
print("The Matthews correlation coefficient is{}".format(MCC))

LABELS = ['Normal', 'Fraud']
conf_matrix = confusion_matrix(yTest, yPred)
plt.figure(figsize =(12, 12))
sns.heatmap(conf_matrix, xticklabels = LABELS,
                        yticklabels = LABELS, annot = True, fmt ="d");
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```

Output-

```
PS C:\Users\pc> & C:/Users/pc/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pc/OneDrive/Desktop/ML in Industries/credit_card_fraud_detection.py"
(284807, 31)
                Time            V1            V2            V3  ...           V27           V28         Amount          Class
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  ...  2.848070e+05  2.848070e+05  284807.000000  284807.000000
mean    94813.859575  1.168375e-15  3.416908e-16 -1.379537e-15  ... -3.660091e-16 -1.227390e-16      88.349619       0.001727
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  ...  4.036325e-01  3.300833e-01     250.120109       0.041527
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01  ... -2.256568e+01 -1.543008e+01       0.000000       0.000000
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01  ... -7.083953e-02 -5.295979e-02       5.600000       0.000000
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01  ...  1.342146e-03  1.124383e-02      22.000000       0.000000
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  ...  9.104512e-02  7.827995e-02      77.165000       0.000000
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  ...  3.161220e+01  3.384781e+01   25691.160000       1.000000

[8 rows x 31 columns]
0.0017304750013189597
Fraud Cases: 492
Valid Transactions: 284315
Amount details of the fraudulent transaction
details of valid transaction
```



EXPERIMENT-6
HYPOTHESIS TESTING

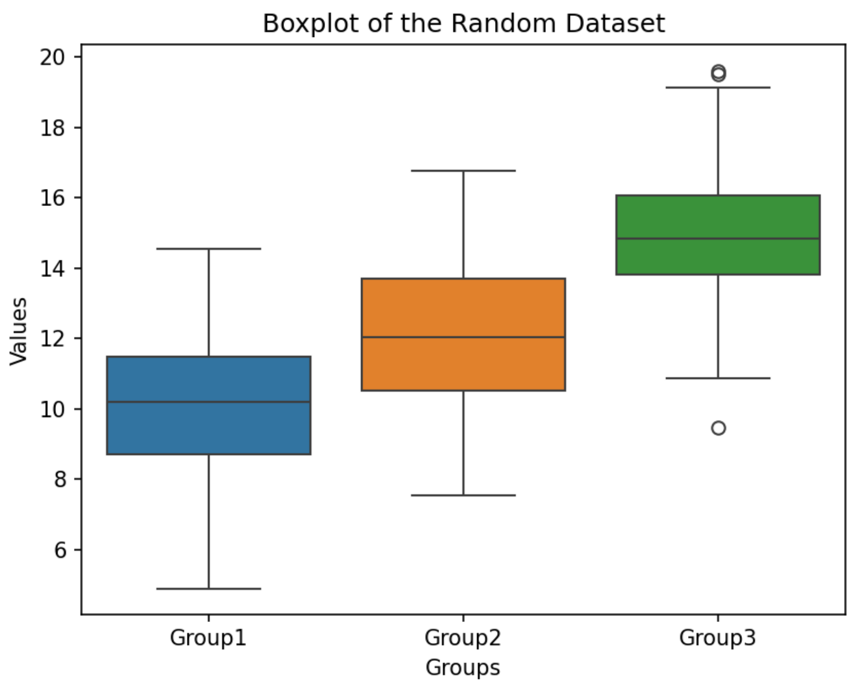
CODE-

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats


np.random.seed(0)
group1 = np.random.normal(loc=10, scale=2, size=100)
group2 = np.random.normal(loc=12, scale=2, size=100)
group3 = np.random.normal(loc=15, scale=2, size=100)


data = pd.DataFrame({
    'Group1': group1,
    'Group2': group2,
    'Group3': group3
})


sns.boxplot(data=data)
plt.title('Boxplot of the Random Dataset')
plt.xlabel('Groups')
```

```python
plt.ylabel('Values')
plt.show()


t_statistic, p_value = stats.ttest_ind(group1, group2)
print("T-Test between Group1 and Group2:")
print("T-statistic:", t_statistic)
print("P-value:", p_value)

z_statistic, p_value = stats.zscore([group1, group2])
print("\nZ-Test between Group1 and Group2:")
print("Z-statistic:", z_statistic[0] - z_statistic[1])
print("P-value:", p_value)


f_statistic, p_value = stats.f_oneway(group1, group2, group3)
print("\nANOVA:")
print("F-statistic:", f_statistic)
print("P-value:", p_value)
```

OUTPUT-

```
PS C:\Users\pc> & C:/Users/pc/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pc/OneDriv
nova.py"
T-Test between Group1 and Group2:
T-statistic: -7.0414273690132765
P-value: 3.059820094513985e-11

Z-Test between Group1 and Group2:
Z-statistic: -2.0
P-value: [ 1. -1. -1. -1. -1.  1. -1.  1.  1.  1.  1.  1. -1.  1.  1.  1.  1.  1.
  1.  1.  1. -1.  1.  1. -1.  1.  1.  1.  1. -1.  1.  1.  1.  1.  1.  1.
  1. -1.  1.  1.  1.  1.  1. -1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1. -1.  1. -1.  1.  1.  1.  1.  1.  1. -1.  1.  1. -1.  1.  1.  1.  1.
  1.  1. -1.  1.  1.  1.  1.  1.  1. -1. -1.  1.  1. -1. -1.  1.  1.  1.
  1.  1.  1. -1.  1.  1.  1.  1.  1.  1.]

ANOVA:
F-statistic: 141.5873093832312
P-value: 6.544281608055403e-44
```



EXPERIMENT-7
Neural network(LSTM) model on sequential dataset

Code-

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Embedding, SpatialDropout1D
from tensorflow.keras.preprocessing.sequence import pad_sequences

max_features = 20000
maxlen = 100
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)

embedding_dim = 128
lstm_units = 64

model = Sequential([
    Embedding(max_features, embedding_dim, input_length=maxlen),
    SpatialDropout1D(0.2),
    LSTM(lstm_units, dropout=0.2, recurrent_dropout=0.2),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

batch_size = 128
epochs = 5
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_data=(x_test, y_test))
```

```
loss, accuracy = model.evaluate(x_test, y_test)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

Output-

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 0s 0us/step
Epoch 1/5
196/196 [==============================] - 97s 478ms/step - loss: 0.4498 - accuracy: 0.7788 - val_loss: 0.3444 - val_accuracy: 0.8502
Epoch 2/5
196/196 [==============================] - 103s 524ms/step - loss: 0.2531 - accuracy: 0.9016 - val_loss: 0.3918 - val_accuracy: 0.8467
Epoch 3/5
196/196 [==============================] - 103s 528ms/step - loss: 0.1755 - accuracy: 0.9350 - val_loss: 0.3758 - val_accuracy: 0.8427
Epoch 4/5
196/196 [==============================] - 102s 519ms/step - loss: 0.1280 - accuracy: 0.9546 - val_loss: 0.4214 - val_accuracy: 0.8352
Epoch 5/5
196/196 [==============================] - 92s 468ms/step - loss: 0.0942 - accuracy: 0.9672 - val_loss: 0.5009 - val_accuracy: 0.8285
782/782 [==============================] - 24s 30ms/step - loss: 0.5009 - accuracy: 0.8285
Test Loss: 0.5008736848831177
Test Accuracy: 0.8285199999809265
```

EXPERIMENT-8
Compare the LSTM(Neural network model)&CNN(Deep learning model)

CODE-

```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from keras.utils import to_categorical

(X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train = X_train / 255.0
X_test = X_test / 255.0

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

def create_sequential_model():
    model = Sequential([
        Flatten(input_shape=(28, 28)),
        Dense(128, activation='relu'),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
    return model
```

```
sequential_model = create_sequential_model()
sequential_history = sequential_model.fit(X_train, y_train,
epochs=10, batch_size=128, validation_data=(X_test, y_test))

def create_cnn_model():
    model = Sequential([
        Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=(28, 28, 1)),
        MaxPooling2D(pool_size=(2, 2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
    return model

X_train_cnn = X_train.reshape(-1, 28, 28, 1)
X_test_cnn = X_test.reshape(-1, 28, 28, 1)

cnn_model = create_cnn_model()
cnn_history = cnn_model.fit(X_train_cnn, y_train, epochs=10,
batch_size=128, validation_data=(X_test_cnn, y_test))

def plot_history(history, title):
    plt.plot(history.history['accuracy'], label='accuracy')
    plt.plot(history.history['val_accuracy'], label='val_accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title(title)
    plt.legend()
```

```
    plt.show()

plot_history(sequential_history, 'Sequential Model')
plot_history(cnn_history, 'CNN Model')
```

OUTPUT-

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
C:\Users\pc\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\reshaping\flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_d
im` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
2024-04-15 10:31:18.078682: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performa
nce-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8295 - loss: 0.6176 - val_accuracy: 0.9454 - val_loss: 0.1879
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9506 - loss: 0.1744 - val_accuracy: 0.9605 - val_loss: 0.1311
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9661 - loss: 0.1156 - val_accuracy: 0.9691 - val_loss: 0.1045
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9749 - loss: 0.0879 - val_accuracy: 0.9705 - val_loss: 0.0958
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9795 - loss: 0.0692 - val_accuracy: 0.9762 - val_loss: 0.0796
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9835 - loss: 0.0559 - val_accuracy: 0.9742 - val_loss: 0.0797
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9876 - loss: 0.0457 - val_accuracy: 0.9791 - val_loss: 0.0709
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9890 - loss: 0.0396 - val_accuracy: 0.9788 - val_loss: 0.0718
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9906 - loss: 0.0329 - val_accuracy: 0.9781 - val_loss: 0.0716
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9930 - loss: 0.0277 - val_accuracy: 0.9765 - val_loss: 0.0703
```

```
C:\Users\pc\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:99: UserWarning: Do not pass an `input_shape`/`i
nput_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 7s 13ms/step - accuracy: 0.8736 - loss: 0.4448 - val_accuracy: 0.9763 - val_loss: 0.0766
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 13ms/step - accuracy: 0.9789 - loss: 0.0747 - val_accuracy: 0.9823 - val_loss: 0.0540
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 13ms/step - accuracy: 0.9862 - loss: 0.0454 - val_accuracy: 0.9820 - val_loss: 0.0491
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 13ms/step - accuracy: 0.9901 - loss: 0.0328 - val_accuracy: 0.9850 - val_loss: 0.0431
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 13ms/step - accuracy: 0.9929 - loss: 0.0238 - val_accuracy: 0.9861 - val_loss: 0.0401
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 13ms/step - accuracy: 0.9947 - loss: 0.0170 - val_accuracy: 0.9849 - val_loss: 0.0454
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 13ms/step - accuracy: 0.9963 - loss: 0.0147 - val_accuracy: 0.9845 - val_loss: 0.0438
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 13ms/step - accuracy: 0.9971 - loss: 0.0100 - val_accuracy: 0.9858 - val_loss: 0.0464
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 13ms/step - accuracy: 0.9978 - loss: 0.0081 - val_accuracy: 0.9846 - val_loss: 0.0479
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 13ms/step - accuracy: 0.9983 - loss: 0.0066 - val_accuracy: 0.9851 - val_loss: 0.0473
```