DEDA Assignment 2:

Individual Assignment

S5714052 Szu Chi Fu

Contents

Section 1: Output the derivative of a function using tensor flow package in python

In this part I create a neural network which used TensorFlow's Keras API, trains it on sample data (x,y)=(1,2), (2,4), (3,6), and then computes the derivative of the model output with respect to the input using tf.GradientTape() function.

```
import tensorflow as tf
# Define the neural network model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=10, activation='relu', input_shape=(1,)),
    tf.keras.layers.Dense(units=1)  # Assuming a regression task with one output
])
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
# Generate input and output sample data
x_train = tf.constant([[1.0], [2.0], [3.0]], dtype=tf.float32)
y_train = tf.constant([[2.0], [4.0], [6.0]], dtype=tf.float32)
# Train the model on the sample data
model.fit(x_train, y_train, epochs=1000, verbose=0)
# Define a function that wraps the model prediction
def f(x):
    return model(x)
# Use TensorFlow's GradientTape to compute the derivative
x_value = tf.constant([[4.0]], dtype=tf.float32)
with tf.GradientTape() as tape:
    tape.watch(x_value)
    y_pred = f(x_value)
# Get the derivative of the output with respect to the input
derivative = tape.gradient(y_pred, x_value)
model.summary()
print("Function Output:", y_pred.numpy())
print("Derivative:", derivative.numpy())
```

The whole process can be divided into five parts:

(1) Define the Neural Network Model:

The neural network model is defined using TensorFlow's Keras API, it consists of two layers, first one dense layer with 10 units and ReLU activation, also for this layer the input shape is set to (1,). The other dense layer with 1 unit which means that the regression task has only one single output.

(2) Compile the model:

Compile the model using Adam optimizer and mean square error as the loss function.

(3) Set up the sample data:

Set up the training dataset x_train and y_train, and let (x._train,y_train)=(1,2), (2,4), (3,6)

(4) Model Training:

Fit the model for 1000 epochs with the function model.fit(x_train, y_train, epochs=1000, verbose=0)

(5) Compute model output and derivative:

Set up the function f(x) to get the prediction of the model, and use tf.GradientTape() function to compute the model's output y_pred for input x_value=4. And further compute the derivative of the model using the function tape.gradient(y_pred, x_value).

After all, we can get the result below:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 10)                20

 dense_1 (Dense)             (None, 1)                 11

=================================================================
Total params: 31 (124.00 Byte)
Trainable params: 31 (124.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Function Output: [[6.3846455]]
Derivative: [[1.2016783]]
```

We can see that there are two dense layers in the model with the output=6.38 and derivative=1.20, finally we get the derivative of the function.

Section 2: Predicting the Ozone Concentration Level by 6 different classification models

I choose the same dataset as our group assignment one, but change the machine learning model from three regression models into six classification models. To find out if classification tools can generate precise solutions to the problem.

1. Introduction

1.1 Necessity

Ozone(O_3), consisting of three oxygen atoms, is a highly reactive gas that can be found in both stratosphere and troposphere. Ozone in the stratosphere is known to be beneficial to humans for blocking the harmful ultraviolet rays from the sun, which is known as the 'ozone layer'. On the other hand, those in the troposphere make our respiratory system

vulnerable, sometimes leading to asthma. Also, they can hinder the growth of vegetation. Moreover, ozone has a strong purifying and sterilizing effect, bacteria and mold in nature cannot reproduce abnormally and maintain a balanced state. It is mainly generated by and VOC(Volatile Organic Compounds). Strong sunlight can decompose and VOC into oxygen and other chemicals, and when it reacts with, they form an ozone.[1]

Predicting ozone levels can offer several advantages, particularly in environmental protection, public health, and meteorology.

For the aspect of environmental protection: (1) Governments set standards for ozone levels to protect air quality, accurate predictions help monitor compliance with these standards, enabling timely interventions to reduce pollution and maintain environmental quality. (2) Ozone levels are often associated with traffic-related emissions. Predictive models can aid urban planners in developing effective traffic management strategies to reduce ozone pollution in congested areas. (3) Predicting ozone levels allows for educational outreach programs to inform the public about the potential health risks associated with high ozone concentrations. This awareness can lead to better individual and community-level actions to reduce pollution.

For the aspect of public health: (1) Ozone is a major component of smog and can have adverse effects on respiratory health. Accurate predictions allow authorities to issue timely warnings, especially for vulnerable populations, and implement measures to mitigate exposure. (2) Accurate predictions facilitate effective emergency response planning. During periods of high ozone levels, resources can be allocated to respond to increased health risks, and emergency services can be prepared for potential respiratory-related incidents.

For the aspect of meteorology: (1) Ozone levels are influenced by various meteorological factors. Predicting ozone levels can provide insights into atmospheric conditions, contributing to more accurate weather forecasting and understanding climate patterns. (2) Ozone levels can impact the amount of solar radiation reaching the Earth's surface. Predictions of ozone levels can be valuable for optimizing the management of solar energy resources.

In summary, predicting ozone levels has broad implications for public health, environmental protection, and meteorology. Such predictive models provide valuable information to governments, environmental agencies, and the public to formulate reasonable measures to improve air quality and reduce adverse impacts on the environment and health.

---

[1] https://www.epa.gov/ozone-pollution-and-your-patients-health/what-ozone

1.2 Goal & Outline

In this assignment, I worked to make a classification model that can classify the ozone concentration level with a number of variables that are presented in the textbook.

To build up the model, I use variables such as temperature, wind, and humidity to predict ozone. First of all, I sort outliers to check whether there are extreme values. Then, use correlation and VIF to delete variables that are not independent. After those works, before building up the model, I split 70% of the data into training dataset and the rest 30% into test dataset and also standardize the data so the outcome can be more precise.
I then build up 6 classification models: logistics regression, random forest classifier, gaussian naive bayes classifier, nearest neighbour classifier, SVM classifier, and gradient boosting classifier. And finally, calculate the accuracy scores and weighted f1 score of each to choose the best model. The details will be shown in this paper later.

2. Methodology & Results

In this part, we will talk more in detail about what we did for our project and the corresponding results we get.

2.1 Data Processing

We used the 330 Los Angeles ozone data measured in 1976 from the textbook『Elements of statistical learning: data mining, inference and prediction. 2nd Edition』. Included variables are Vandenberg 500mb height (vh), wind speed in mph (wind), humidity, Sandburg AFT Temperature (temp), Inversion Base Height (ibh), Daggot Pressure Gradient (dpg), Inversion Base Temperature (ibt), visibility (vis).[2] It is worth pointing out that the maximum value of ibh and the minimum value of wind, humidity are fixed as 5000m and 0 respectively. This is based on the property of each variable; the minimum of the wind speed is zero since it is an absolute value, humidity is recorded as percent and the maximum height of the inversion base is 5km.

2.1.1 Sorting the Outliers

In this part, we first calculated both standard deviation and mean of each variable and transformed the data into normal distribution by the formula below:

---

[2] https://hastie.su.domains/ElemStatLearn/datasets/LAozone.info.txt

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

After the transformation, we calculate new standard deviation and mean of each variable, and change the data that are out of the range ($m-3\sigma$,$m+3\sigma$) into $m-3\sigma$ if it is smaller than $m-3\sigma$, and $m+3\sigma$ if it is larger than $m+3\sigma$.

Our code is below:

```
# Sort Outliers: by deleting the data >mean+3 standard deviation and <mean-3 standard
deviation
def sort_outlier(X):
  # Calculate standard deviation and mean
   temp_std = np.std(X)
   temp_mu = np.mean(X)
 # Transform our data into normal distribution
   X_norm_dist = np.e**(-0.5*((X-temp_mu)/temp_std)**2)/(temp_std*np.sqrt(2*np.pi))
 # Calculate new standard deviation and mean for the normal distribution
   std = np.std(X_norm_dist)
   mu = np.mean(X_norm_dist)
 # Check if there are any extreme values and add it into outliers: data outside the range
mean+-3std
   outliers = []
   for i in range(len(X)):
      if X_norm_dist[i] < mu - std*3 or X_norm_dist[i] > mu + std*3:
         outliers.append(i)
   return outliers
# Print the index of the outliers
variables = [oz, vh, wind, humidity, temp, ibh, dpg, ibt, vis]
variables_name = ['oz', 'vh', 'wind', 'humidity', 'temp', 'ibh', 'dpg', 'ibt', 'vis']
for i in range(len(variables)):
   print(f'{variables_name[i]}: {sort_outlier(variables[i])}')
```

The result we get is shown below. Since after the transformation every data was in between the range of ($m-3\sigma$,$m+3\sigma$), there were no outliers.

```
oz: []
vh: []
wind: []
humidity: []
temp: []
ibh: []
dpg: []
ibt: []
vis: []
```

2.1.2 Choosing the Variables

Next, to proceed with regression, we should check if variables are not correlated to each other. Here, we used the correlation coefficient and the scatter plots between the two variables.

For the calculation of correlation coefficient, we divide variables into four groups by their value of correlation. Our code for the correlation coefficient is below:

```
# Define function Corr(x,y) to calculate the correlation between two variables
def Corr(x,y):
    x_mean = np.mean(x)
    y_mean = np.mean(y)
    x_std = np.std(x)
    y_std = np.std(y)
    corr = np.mean((x-x_mean)*(y-y_mean))/(x_std*y_std)
    return corr
# Create lists to store pairs of variables: ignore->corr<0.3 / weak->0.3<=corr<0.5 /
medium->0.5<=corr<0.7 / strong->0.7<=corr<1.0
ignore=[]
ignore_list=[]
weak_corr=[]
weak_list=[]
med_corr=[]
med_list=[]
strong_corr=[]
strong_list=[]
# Calculate the correlation value between every two variables and put them into the corresponding
list
for i in range(len(variables)):
    for j in range(i+1, len(variables)):
        corr_val = np.abs(Corr(variables[i], variables[j]))
        if corr_val < 0.3:
            ignore.append(corr_val)
            ignore_list.append([variables_name[i], variables_name[j]])
        elif 0.3 <= corr_val < 0.5:
            weak_corr.append(corr_val)
            weak_list.append([variables_name[i], variables_name[j]])
        elif 0.5 <= corr_val < 0.7:
            med_corr.append(corr_val)
            med_list.append([variables_name[i], variables_name[j]])
        elif 0.7 <= corr_val:
            strong_corr.append(corr_val)
            strong_list.append([variables_name[i], variables_name[j]])
# Print out the results
print(f'Can be ignored: {len(ignore)}')
for i in range(len(ignore_list)):
    print(f'{ignore_list[i]}: {ignore[i]}')
print('\n')
print(f'Weakly correlated: {len(weak_corr)}')
for i in range(len(weak_list)):
```

```
        print(f'{weak_list[i]}: {weak_corr[i]}')
print('\n')
print(f'Medium correlated: {len(med_corr)}')
for i in range(len(med_list)):
    print(f'{med_list[i]}: {med_corr[i]}')
print('\n')
print(f'Strongly correlated: {len(strong_corr)}')
for i in range(len(strong_list)):
    print(f'{strong_list[i]}: {strong_corr[i]}')
print('\n')
```

Then we gets the result of the table below:

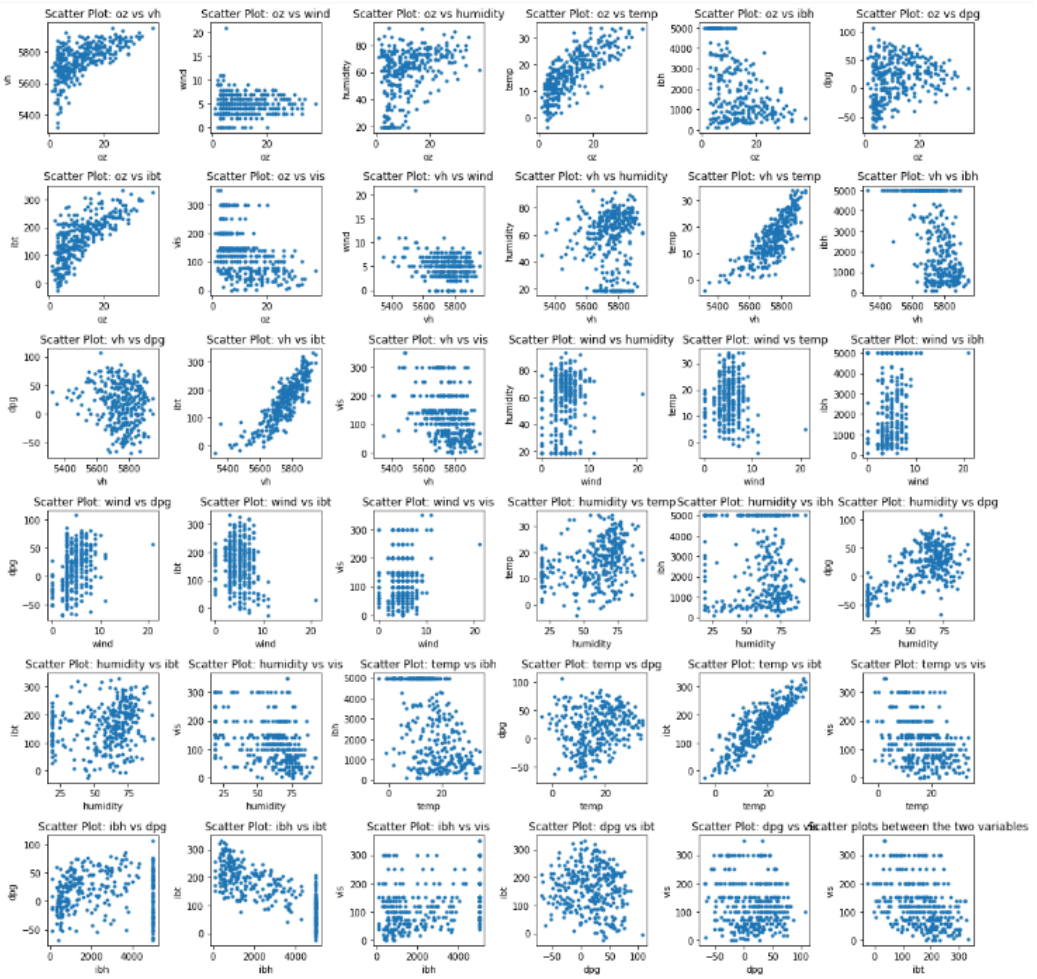
	Ozone	Wind	vh	dpg	Humidity	Temp	ibh	ibt	vis
Ozone	1								
Wind	0.0134	1							
vh	0.6073	0.2436	1						
dpg	0.2140	0.3357	0.1480	1					
Humidity	0.4492	0.2102	0.0744	0.6477	1				
Temp	0.7807	0.0032	0.8080	0.1892	0.3404	1			
ibh	0.5895	0.2065	0.5048	0.0370	0.2423	0.5326	1		
ibt	0.7455	0.1795	0.8520	0.0950	0.2036	0.8647	0.7769	1	
vis	0.4409	0.1472	0.3600	0.1258	0.4010	0.3877	0.3866	0.4223	1

We then check the scatter plots of the variables with the code below:

```
# Plotting the scatter plots to help determine the relationship between variables
from itertools import combinations
# Creating space for subplots
fig, axes = plt.subplots(nrows=6, ncols=6, figsize=(15, 15))
combinations_list = list(combinations(range(len(variables)), 2))
# Plot all the scatter plots into the space
for idx, (i, j) in enumerate(combinations_list):
    row = idx // 6
    col = idx % 6
    x, y = variables[i], variables[j]
    axes[row, col].plot(x, y, '.')
    axes[row, col].set_title(f'Scatter Plot: {variables_name[i]} vs {variables_name[j]}')
    axes[row, col].set_xlabel(variables_name[i])
    axes[row, col].set_ylabel(variables_name[j])
```

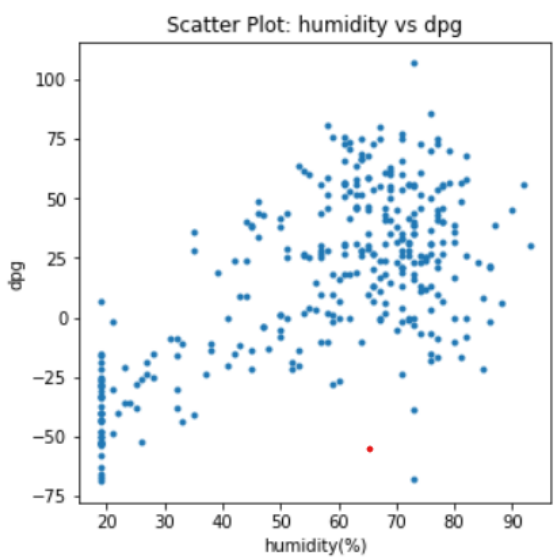
```
# To make sure the layout of the plots are fine
plt.tight_layout()
```

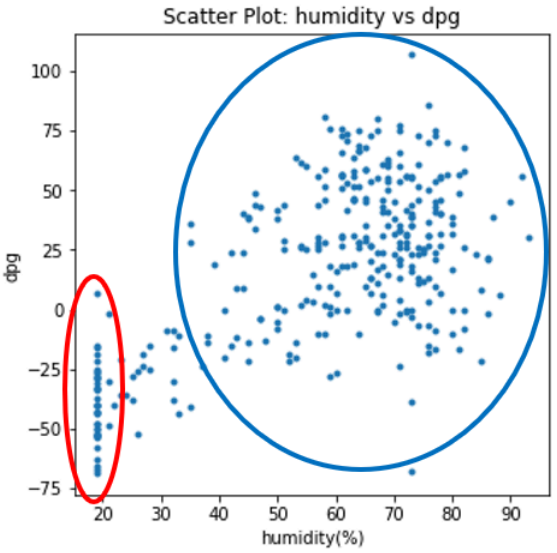
And get the plots below:



Since ozone should be our target variable, we focused on the correlation values between the other eight variables. To start with, by looking at 'strongly correlated variables' and their scatter plots, we can conclude that we should get rid of one of the variables between [vh, temp], [vh, ibt], [temp, ibt] and [ibh, ibt] since we can find a strong linear relationship between the two variables. For the first set, [vh] and [temp] , we decided to remove [vh] since [temp] is more related to [ozone]. Because we remove [vh], the next set we should consider is [temp] and [ibt]. We still kept the [temp] which shows a slightly stronger relation with [ozone] than [ibt].

Secondly, we looked at the 'medium-correlated variables', which are [humidity, dpg], [temp, ibh](we do not consider [vh, ibh] since we get rid of the variable [vh]). To begin with, we concluded that we can see the relationship between [humidity, dpg] as independent.





They might seem to have a linear relationship, but when we get rid of the line-plotted dots(red circle), remaining dots are plotted as a round circle(blue circle), which shows a weak relationship between the two variables. Therefore, we finish our variable selection.

After all, we double checked the multicollinearity of our data by VIF index, the formula of the index is below:

$$VIF_i = \frac{1}{1 - R_i^2}$$

If the index of the variable exceeds 10, then we will delete that variable to ensure that there is no multicollinearity between the variables. The code is below:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
import pandas as pd
data = {
    'wind': wind,
    'humidity': humidity,
    'temp': temp,
    'ibh': ibh,
    'dpg': dpg,
    'vis': vis,
}
# Creating a DataFrame from the data
df = pd.DataFrame(data)
# Adding a constant term to the DataFrame for linear regression
data_frame_with_const = sm.add_constant(df)
# Define function to calculate Variance Inflation Factor
def calculate_vif(data_frame):
```

```
# Create a DataFrame for vif_data, store variable names in "Variable" and calculate "VIF" for corresponding variables
    vif_data = pd.DataFrame()
    vif_data["Variable"] = data_frame.columns
    vif_data["VIF"] = [variance_inflation_factor(data_frame.values, i) for i in range(data_frame.shape[1])]
    return vif_data
# Calculating VIF for our DataFrame with the constant term
vif_result = calculate_vif(data_frame_with_const)
print(vif_result)
```

Since there are no variables that have VIF exceeding 10, we finish all of our steps in data processing.

	Variable	VIF
0	const	40.432918
1	wind	1.217932
2	humidity	2.244617
3	temp	1.559482
4	ibh	1.612055
5	dpg	1.975914
6	vis	1.434881

2.2 Building Models

Before building up all the models, I divide the data into training data and test data. I separate 30% of our data into a test dataset and the rest remains as the training dataset. Since we are dealing with a classification problem, I divide ozone into stages according to different concentration levels, for oz=0~9 I set them to 0; for oz=10~19 set them to 1, and set the left into 2.

I also standardize the input data x_train and x_test by using StandardScaler package, so the accuracy of the model will perform better. Therefore, we get x_train_scaled and x_test_scaled as our new inputs. After all the preparation, we started model training.

Codes for this part is below:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
# Set oz into different classes by their ozone concentration level
oz = [0 if value < 10 else 1 if 10 < value < 20 else 2 for value in oz]
y = oz
# Split 30% of the data in to test dataset, else to training dataset
X = np.vstack([wind, humidity, temp, ibh, dpg, vis]).T
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.3, random_state = 42)
# Standardize the inputs: x_train and x_test
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

During the process of building up every model, some steps are repeated:

(1) Model build and fit:

Build up the model first, then use x_train_scaled as the input and y_train as the output to train the model.

(2) Get the prediction of the model:

Use the function model.predict(x_test_scaled) to get the model predicted outputs.

(3) Calculate model accuracy

Compare the model predicted outputs y_pred with the true outputs y_test. The accuracy score is calculated by the formula below:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

And the classification report shows three different indicators of the model performance: precision, recall, and f1-score. Precision is the ability of the classifier not to label as positive a sample that is negative; Recall is the ability of the classifier to find all the positive samples; F1-Score is the harmonic mean of precision and recall, providing a balance between the two metrics. The formulas of these indexes are also shown below:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

(4) Create scatter plots

Finally, create a 3*5 space and draw scatter plots for x,y axis=all pairs of x variables, and show the classes of y by different colors.

The codes of each model will be shown below in the related section, and I will focus on explaining the methodology of the models in those sections.

2.2.1 Logistics Regression

Logistic regression is a process of modeling the probability of a discrete outcome given an input variable. It predicts probabilities between 0 and 1, providing the likelihood of belonging to a particular class. The formula of the model:

$$P(Y = i) = \frac{e^{\theta_i^T x}}{\sum_{j=1}^{K} e^{\theta_j^T x}}$$

This method is simple to understand and easy to interpret, also with a good efficiency and the low risk of overfitting. But since it assumes a linear relationship between input features and the log odds, it may not capture complex relationships in the data. At the same time, outliers can seriously influence the model parameters, further impacting the performance.

My code for logistics regression:

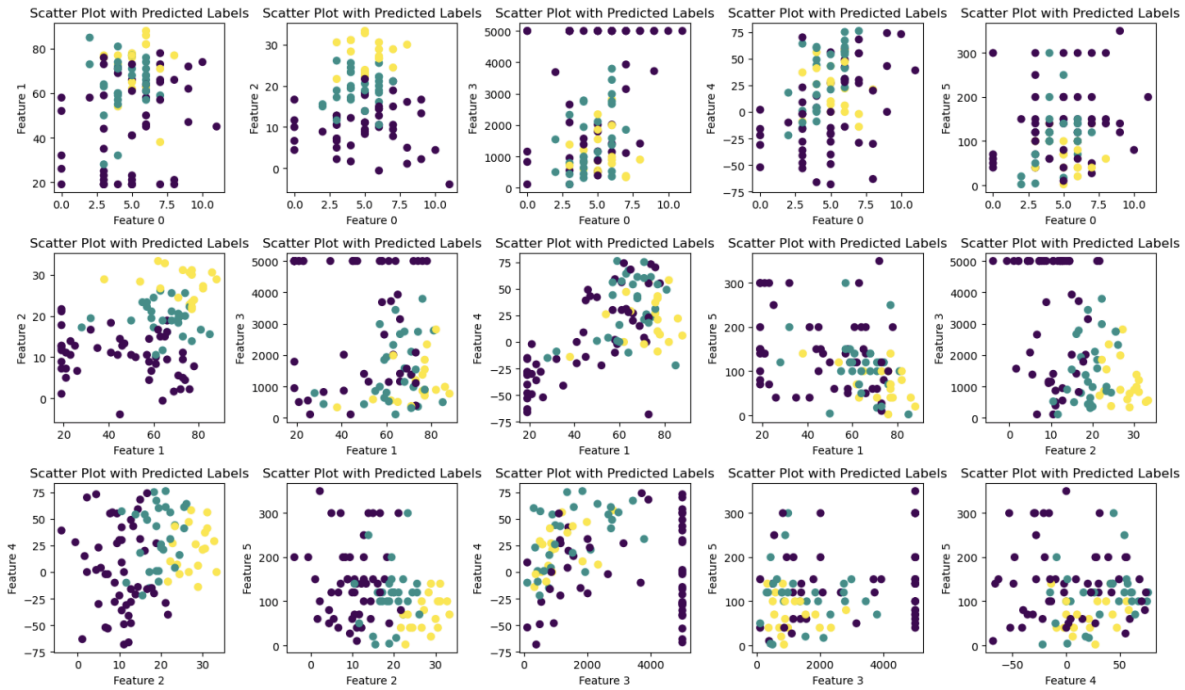
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,f1_score
# Create and train the logistic regression model
model = LogisticRegression(max_iter=1000)
lr_classification = model.fit(X_train_scaled, y_train)
# Make predictions on the test set
y_pred = model.predict(X_test_scaled)
# Evaluate the model
lr_accuracy = accuracy_score(y_test, y_pred)
lr_report = classification_report(y_test, y_pred)
lr_f1score = f1_score(y_test, y_pred,average='weighted')
print(f"Accuracy: {lr_accuracy:.2f}")
print("Classification Report:\n", lr_report)
# Create scatter plots
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(15, 9))
idx = 0
for i in range(5):
    for j in range(i+1, 6, 1):
        row = idx//5
        col = idx%5
        ax = axes[row, col]
        # Scatter plot on the specific subplot
        scatter = ax.scatter(X_test[:, i], X_test[:, j], c=y_pred, s=40, cmap='viridis')
        ax.set_xlabel(f'Feature {i}')
        ax.set_ylabel(f'Feature {j}')
        ax.set_title('Scatter Plot with Predicted Labels')
        idx=idx+1
# To make sure the layout of the plots are fine
plt.tight_layout()
plt.show()
```

And the result I get is below:

```
Accuracy: 0.67
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.85      0.83        48
           1       0.48      0.45      0.47        31
           2       0.58      0.55      0.56        20

    accuracy                           0.67        99
   macro avg       0.62      0.62      0.62        99
weighted avg       0.66      0.67      0.66        99
```



2.2.2 Random Forest Classifier

A Random Forest Classifier is an ensemble learning algorithm, by combining the predictions of multiple models it can improve overall performance and robustness. It consists of a collection of decision trees and combines multiple decision trees to create an accurate model. At each split a random subset of features is considered, the randomization helps prevent overfitting.

In my code, I create 1000 trees in the forest, the code for random forest classifier is below:

```
from sklearn.ensemble import RandomForestClassifier
# Create and train the random forest model
model = RandomForestClassifier(n_estimators=1000)
rf_classification = model.fit(X_train_scaled, y_train)
# Make predictions on the test set
y_pred = model.predict(X_test_scaled)
# Evaluate the model
rf_accuracy = accuracy_score(y_test, y_pred)
rf_report = classification_report(y_test, y_pred)
```

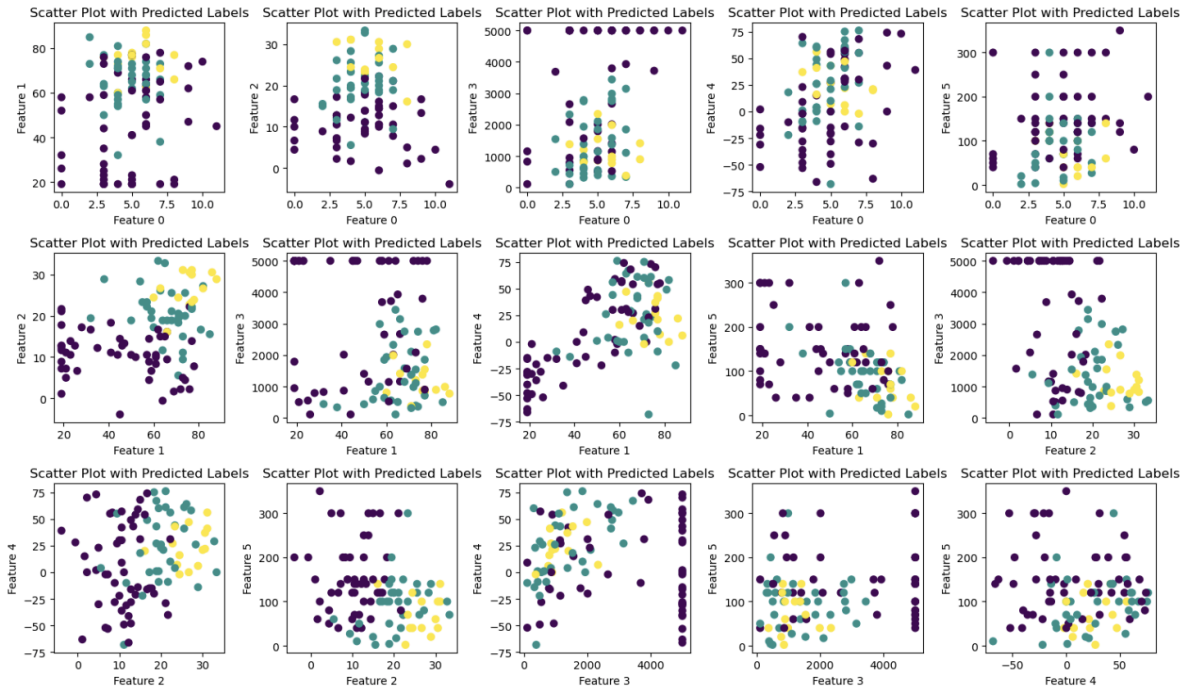
```
rf_f1score = f1_score(y_test, y_pred,average='weighted')
print(f"Accuracy: {rf_accuracy:.2f}")
print("Classification Report:\n", rf_report)
# Create scatter plots
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(15, 9))
idx = 0
for i in range(5):
    for j in range(i+1, 6, 1):
        row = idx//5
        col = idx%5
        ax = axes[row, col]
        # Scatter plot on the specific subplot
        scatter = ax.scatter(X_test[:, i], X_test[:, j], c=y_pred, s=40, cmap='viridis')
        ax.set_xlabel(f'Feature {i}')
        ax.set_ylabel(f'Feature {j}')
        ax.set_title('Scatter Plot with Predicted Labels')
        idx=idx+1
# To make sure the layout of the plots are fine
plt.tight_layout()
plt.show()
```

And the result I get is below:

```
Accuracy: 0.67
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.85      0.83        48
           1       0.53      0.58      0.55        31
           2       0.50      0.35      0.41        20

    accuracy                           0.67        99
   macro avg       0.61      0.59      0.60        99
weighted avg       0.66      0.67      0.66        99
```



2.2.3 Gaussian Naive Bayes Classifier

The Gaussian Naive Bayes Classifier is a probabilistic classification algorithm based on Bayes' theorem. It is a variant of the Naive Bayes algorithm and is specifically designed for datasets where the features are continuous and assumed to follow a Gaussian (normal) distribution. It's computationally efficient and easy to implement and can perform well even with a small dataset.

My code for gaussian naive bayes classifier:

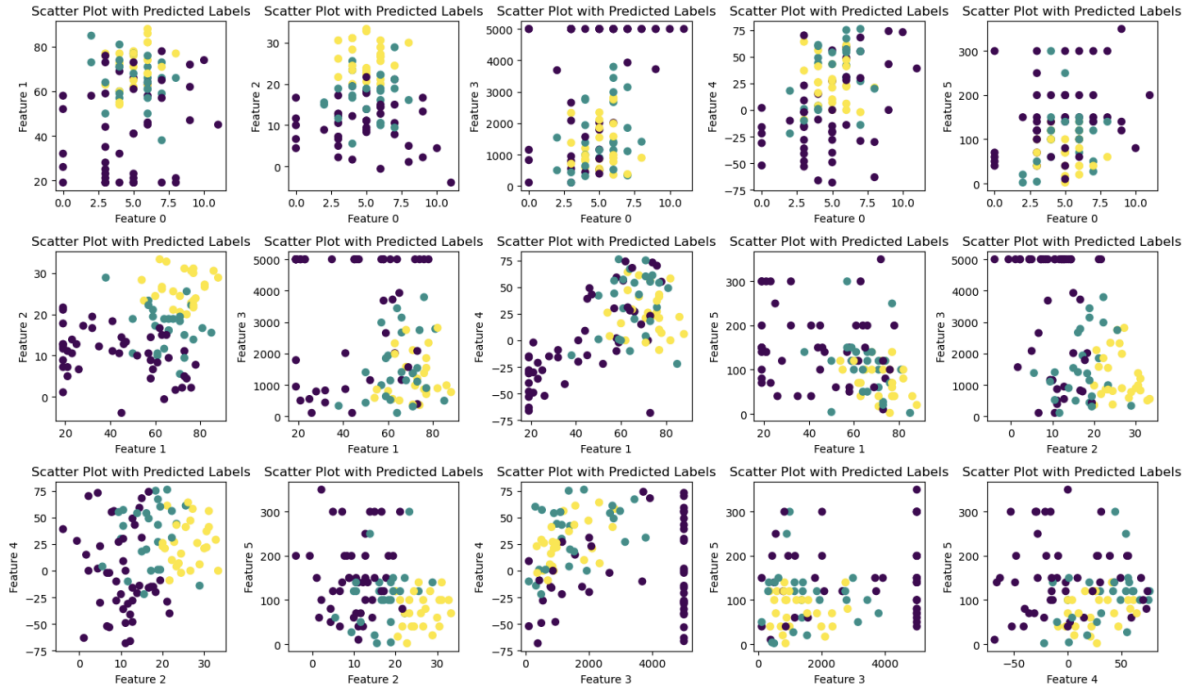
```
from sklearn.naive_bayes import GaussianNB
# Create and train the Gaussian Naive Bayes model
model = GaussianNB()
gnb_classification = model.fit(X_train_scaled, y_train)
# Make predictions on the test set
y_pred = model.predict(X_test_scaled)
# Evaluate the model
gnb_accuracy = accuracy_score(y_test, y_pred)
gnb_report = classification_report(y_test, y_pred)
gnb_f1score = f1_score(y_test, y_pred,average='weighted')
print(f"Accuracy: {gnb_accuracy:.2f}")
print("Classification Report:\n", gnb_report)
# Create scatter plots
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(15, 9))
idx = 0
for i in range(5):
    for j in range(i+1, 6, 1):
        row = idx//5
        col = idx%5
        ax = axes[row, col]
        # Scatter plot on the specific subplot
        scatter = ax.scatter(X_test[:, i], X_test[:, j], c=y_pred, s=40, cmap='viridis')
        ax.set_xlabel(f'Feature {i}')
        ax.set_ylabel(f'Feature {j}')
        ax.set_title('Scatter Plot with Predicted Labels')
        idx=idx+1
# To make sure the layout of the plots are fine
plt.tight_layout()
plt.show()
```

And the result I get is below:

```
Accuracy: 0.70
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.85      0.86        48
           1       0.54      0.45      0.49        31
           2       0.54      0.70      0.61        20

    accuracy                           0.70        99
   macro avg       0.65      0.67      0.65        99
weighted avg       0.70      0.70      0.70        99
```



2.2.4 Nearest Neighbours Classifier

Nearest Neighbors Classifier is an instance based learning algorithm, it doesn't build a model during the training process. Instead it memorizes the training instances and makes predictions based on the proximity of new instances to the training data. The class label is determined by a majority vote among the k neighbors. The number of neighbors k will be the hyperparameter that needs to be tuned. Smaller values of k can be more flexible, but noisy. Larger values of k can smooth out noise but may miss local patterns.

My code for logistics regression:

```
from sklearn import neighbors
# Create and train the logistic regression model
model = neighbors.KNeighborsClassifier()
nn_classification = model.fit(X_train_scaled, y_train)
# Make predictions on the test set
y_pred = model.predict(X_test_scaled)
# Evaluate the model
nn_accuracy = accuracy_score(y_test, y_pred)
```

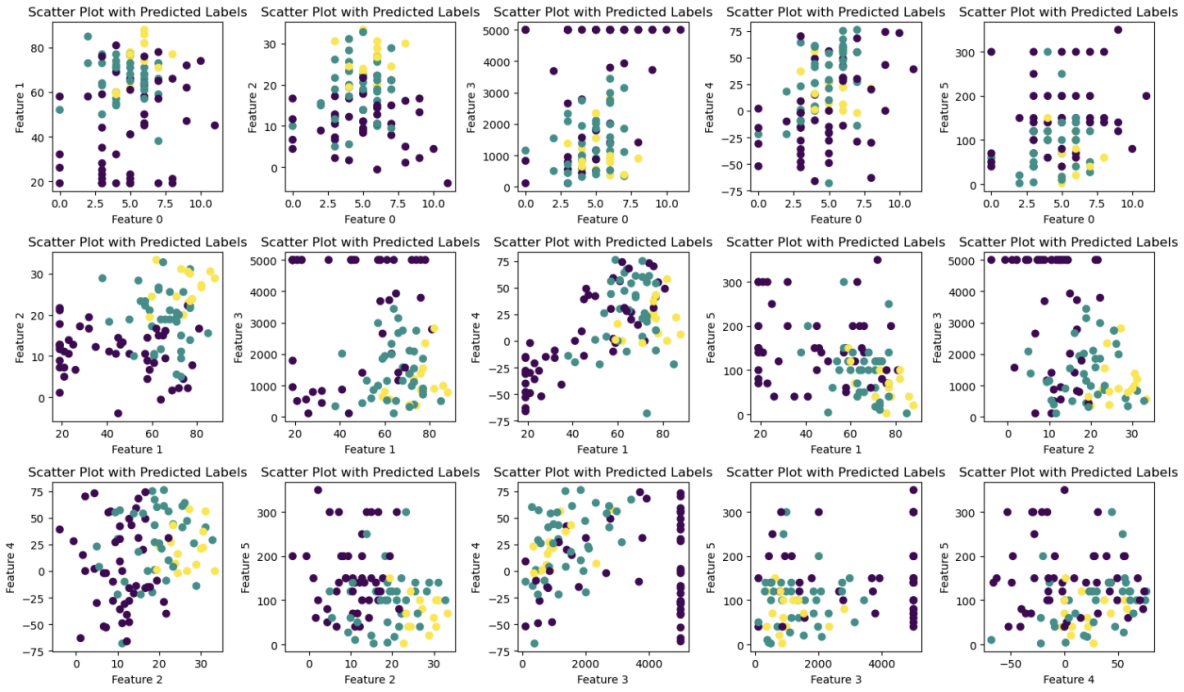
```
nn_report = classification_report(y_test, y_pred)
nn_f1score = f1_score(y_test, y_pred,average='weighted')
print(f"Accuracy: {nn_accuracy:.2f}")
print("Classification Report:\n", nn_report)
# Create scatter plots
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(15, 9))
idx = 0
for i in range(5):
    for j in range(i+1, 6, 1):
        row = idx//5
        col = idx%5
        ax = axes[row, col]
        # Scatter plot on the specific subplot
        scatter = ax.scatter(X_test[:, i], X_test[:, j], c=y_pred, s=40, cmap='viridis')
        ax.set_xlabel(f'Feature {i}')
        ax.set_ylabel(f'Feature {j}')
        ax.set_title('Scatter Plot with Predicted Labels')
        idx=idx+1
# To make sure the layout of the plots are fine
plt.tight_layout()
plt.show()
```

And the result I get is below:

```
Accuracy: 0.63
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.79      0.80        48
           1       0.46      0.55      0.50        31
           2       0.47      0.35      0.40        20

    accuracy                           0.63        99
   macro avg       0.58      0.56      0.57        99
weighted avg       0.63      0.63      0.63        99
```



2.2.5 SVM Classifier

A SVM classifier is a supervised machine learning algorithm used for both classification and regression tasks. SVM is particularly effective in high dimensional spaces and is well suited for cases where the decision boundary between classes is non linear or complex. The primary objective of SVM is to find a hyperplane that best separates data points belonging to different classes. The margin concept of SVM helps prevent overfitting, especially with a proper choice of the regularization parameter.

My code for logistics regression:

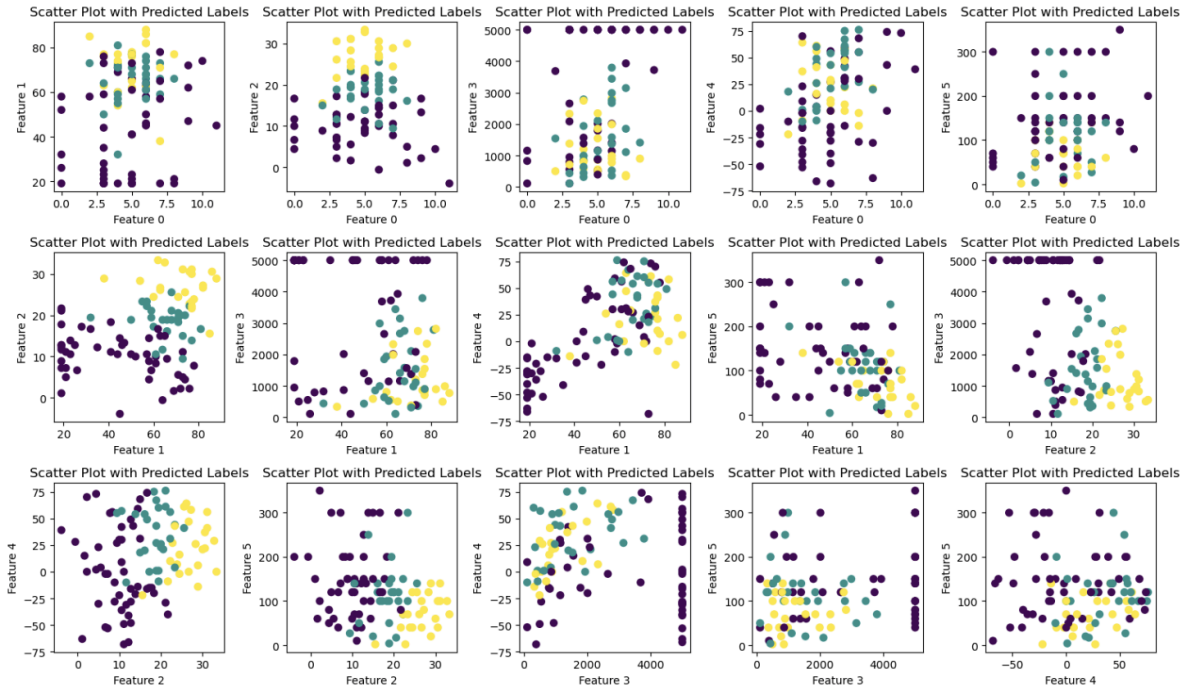
```
from sklearn import svm
# Create and train the SVM model
model = svm.SVC(kernel="linear")
svm_classification = model.fit(X_train_scaled, y_train)
# Make predictions on the test set
y_pred = model.predict(X_test_scaled)
# Evaluate the model
svm_accuracy = accuracy_score(y_test, y_pred)
svm_report = classification_report(y_test, y_pred)
svm_f1score = f1_score(y_test, y_pred,average='weighted')
print(f"Accuracy: {svm_accuracy:.2f}")
print("Classification Report:\n", svm_report)
# Create scatter plots
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(15, 9))
idx = 0
for i in range(5):
    for j in range(i+1, 6, 1):
        row = idx//5
        col = idx%5
        ax = axes[row, col]
        # Scatter plot on the specific subplot
        scatter = ax.scatter(X_test[:, i], X_test[:, j], c=y_pred, s=40, cmap='viridis')
        ax.set_xlabel(f'Feature {i}')
        ax.set_ylabel(f'Feature {j}')
        ax.set_title('Scatter Plot with Predicted Labels')
        idx=idx+1
# To make sure the layout of the plots are fine
plt.tight_layout()
plt.show()
```

And the result I get is below:

```
Accuracy: 0.68
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.85      0.85        48
           1       0.52      0.45      0.48        31
           2       0.52      0.60      0.56        20

    accuracy                           0.68        99
   macro avg       0.63      0.64      0.63        99
weighted avg       0.67      0.68      0.67        99
```



2.2.6 Gradient Boosting Classifier

Gradient Boosting Classifier is an ensemble machine learning technique used for classification tasks. It belongs to the family of boosting algorithms. The primary idea behind gradient boosting is to combine the predictions of multiple weak learners to create a strong predictive model. Main objective of gradient boosting is to minimize a loss function. It starts with an initial model and fits subsequent models to the negative gradient of the loss function with respect to the predictions of the existing model. Each new model is trained to minimize the error, moving in the direction of the negative gradient of the loss function.

My code for logistics regression:

```
from sklearn.ensemble import GradientBoostingClassifier
# Create and train the Gradient Boosting model
model = GradientBoostingClassifier()
gb_classification = model.fit(X_train_scaled, y_train)
# Make predictions on the test set
y_pred = model.predict(X_test_scaled)
# Evaluate the model
```

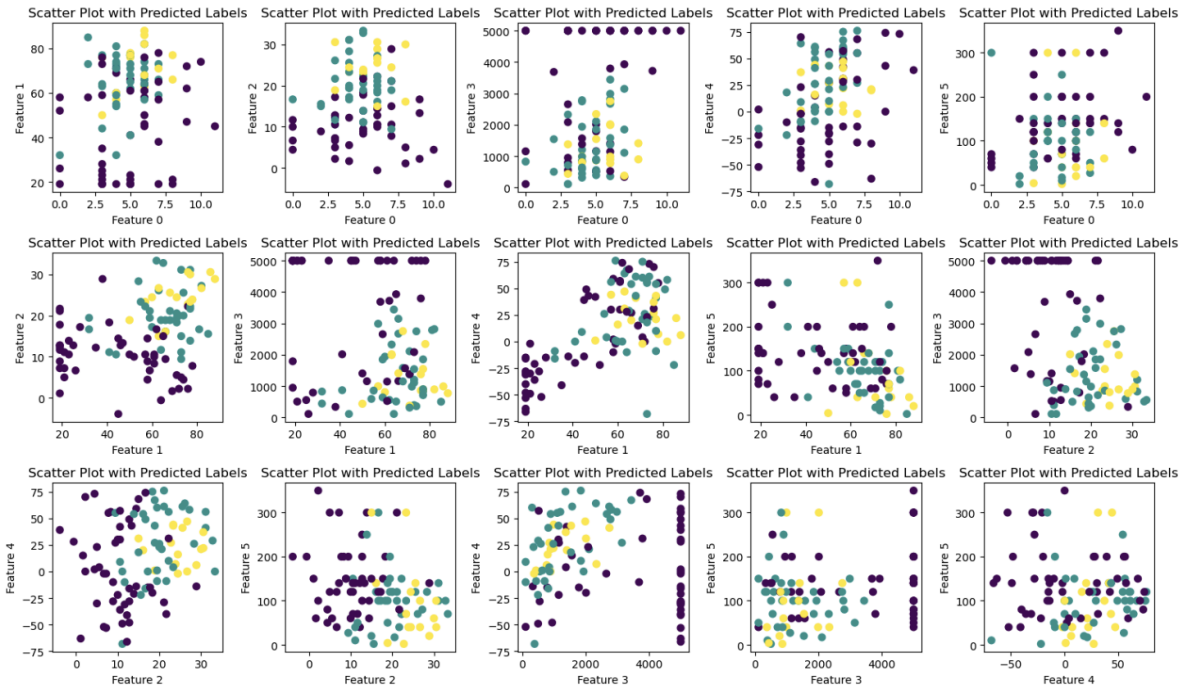
```
gb_accuracy = accuracy_score(y_test, y_pred)
gb_report = classification_report(y_test, y_pred)
gb_f1score = f1_score(y_test, y_pred,average='weighted')
print(f"Accuracy: {gb_accuracy:.2f}")
print("Classification Report:\n", gb_report)
# Create scatter plots
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(15, 9))
idx = 0
for i in range(5):
    for j in range(i+1, 6, 1):
        row = idx//5
        col = idx%5
        ax = axes[row, col]
        # Scatter plot on the specific subplot
        scatter = ax.scatter(X_test[:, i], X_test[:, j], c=y_pred, s=40, cmap='viridis')
        ax.set_xlabel(f'Feature {i}')
        ax.set_ylabel(f'Feature {j}')
        ax.set_title('Scatter Plot with Predicted Labels')
        idx=idx+1
# To make sure the layout of the plots are fine
plt.tight_layout()
plt.show()
```

And the result I get is below:

```
Accuracy: 0.59
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.79      0.80        48
           1       0.42      0.48      0.45        31
           2       0.31      0.25      0.28        20

    accuracy                           0.59        99
   macro avg       0.51      0.51      0.51        99
weighted avg       0.59      0.59      0.58        99
```



2.3 Model Selection

We use accuracy score and weighted f1-score to conduct model selection.

```
# Choosing the best model with the largest accuracy score
best_model_accuracy = max([(lr_accuracy, 'Logistics regression'), (rf_accuracy, 'Random Forest Classifier'), (gnb_accuracy, 'Gaussian Naive Bayes Classifier'), (nn_accuracy, 'Nearest Neighbours Classification'), (svm_accuracy, 'SVM Classification'), (gb_accuracy, 'Gradient Boosting Classifier')])
print(f"Best Model evaluating by accuracy: {best_model_accuracy[1]} with accuracy score = {best_model_accuracy[0]}")
# Choosing the best model with the largest weighted f1-score
best_model_f1score = max([(lr_f1score, 'Logistics regression'), (rf_f1score, 'Random Forest Classifier'), (gnb_f1score, 'Gaussian Naive Bayes Classifier'), (nn_f1score, 'Nearest Neighbours Classification'), (svm_f1score, 'SVM Classification'), (gb_f1score, 'Gradient Boosting Classifier')])
print(f"Best Model evaluating by f1-score: {best_model_f1score[1]} with f1 score = {best_model_f1score[0]}")
```

By this code, we can get the table below:

	Accuracy	f1 score
Logistics Regression	0.67	0.66
Random Forest Classifier	0.66	0.65
Gaussian Naive Bayes Classifier	0.70	0.70
Nearest Neighbours Classifier	0.63	0.63
SVM Classifier	0.68	0.67
Gradient Boosting Classifier	0.59	0.58

By the results we get, we can conclude that the Gaussian Naive Bayes Classifier Model performs slightly better than the other five.

3. Conclusion

3.1 Discussion

In this project, I try to predict the concentration level of ozone by the input variables wind, humidity, temp, ibh, dpg, and vis. I tested six different classification models and finally found out the gaussian naive bayes classifier performs the best with the accuracy score 0.7 and f1 score 0.7. But I also observed that although this model performs better, the effect is not significant. There may be few possible characteristics of the dataset.

The dataset may have a moderate level of complexity, making it interpretable by various algorithms. If the relationships between features and the target variable are not really complex, multiple models might converge to similar accuracy levels. The features in

the dataset may not provide sufficient discriminatory information to strongly favor one model over another. If the feature space does not have distinctive patterns, different algorithms may achieve comparable performance.

Or maybe the chosen models cover a spectrum of complexity levels. If the dataset does not require highly complex models, simpler models might be sufficient, leading to similar accuracy scores. If the dataset contains noise or irrelevant features, more complex models may not necessarily perform better. Noise could overshadow true patterns in the data, impacting the models uniformly. Also models might be underfitting or overfitting, leading to similar accuracy scores.

3.2 Proposal for future research

For future research on this topic, I think there can be more input variables, we should consider if there are any other factors that affect ozone concentration level. So we can further improve our model's accuracy and performance.

We can also consider that if there is a different method that we can build up the problem and model(ex: regression, classification). Maybe there is a different method that fits the task better.