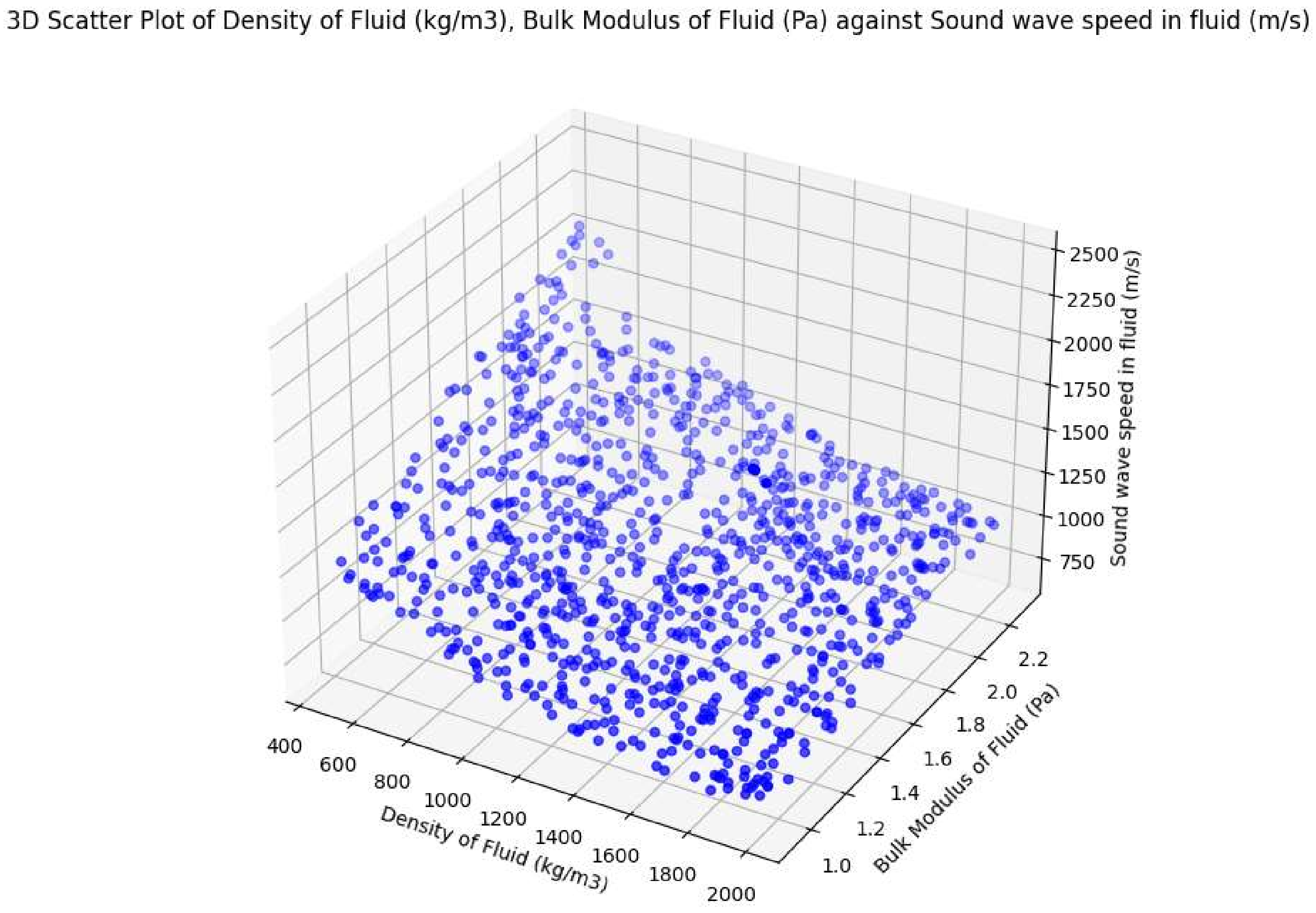
```
from google.colab import drive
drive.mount('/content/gdrive')

     Mounted at /content/gdrive
```

```
dataset_path = '/content/gdrive/My Drive/ME5201_assignment/Training Dataset ME5201.xlsx'
```

```
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Load your dataset
data = pd.read_excel(dataset_path)
x1_col = 'Density of Fluid (kg/m3)'
x2_col = 'Bulk Modulus of Fluid (Pa)'
y_col = 'Sound wave speed in fluid (m/s)'

# 3D Scatter plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(data[x1_col], data[x2_col], data[y_col], c='blue', marker='o')

# Set labels
ax.set_xlabel(x1_col)
ax.set_ylabel(x2_col)
ax.set_zlabel(y_col)

plt.title(f'3D Scatter Plot of {x1_col}, {x2_col} against {y_col}')
plt.show()
```



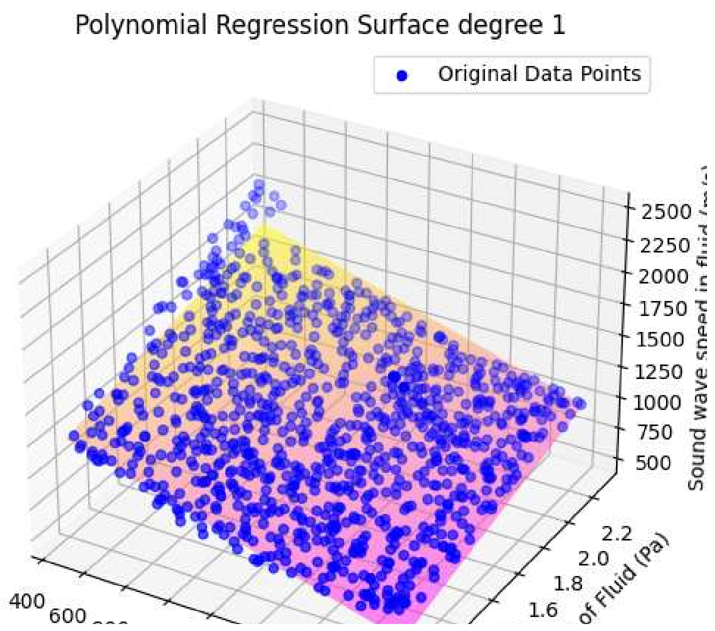
```
# for linear regression degree 1

from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

x1_col = data['Density of Fluid (kg/m3)'].values
x2_col = data['Bulk Modulus of Fluid (Pa)'].values
y_col = data['Sound wave speed in fluid (m/s)'].values

# # Reshape the data to have a single feature
x1_col = x1_col.reshape(-1, 1)
x2_col = x2_col.reshape(-1, 1)
y_col = y_col.reshape(-1, 1)

# Combine the features into a single array
features = np.concatenate((x1_col, x2_col), axis=1)

# Create polynomial features
poly_features = PolynomialFeatures(degree=1)
features_poly = poly_features.fit_transform(features)

# Create and fit the linear regression model
model = LinearRegression()
model.fit(features_poly, y_col)

# Print the equation
equation = f"z = {model.intercept_[0]}"

for idx, coef in enumerate(model.coef_[0]):
    equation += f" + {coef}*x^({poly_features.powers_[idx][0]})*y^({poly_features.powers_[idx][1]})"

print("Equation:", equation)

# Predict the target variable
predicted_y_col = model.predict(features_poly)
print(model)
# Calculate evaluation metrics
mse = mean_squared_error(y_col, predicted_y_col)
rmse = np.sqrt(mse)
r2 = r2_score(y_col, predicted_y_col)

print("Mean Squared Error (MSE):     :", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared (R2)                :", r2)
print()
```

```
Equation: z = 1192.070269671431 + 0.0*x^(0)*y^(0) + -0.49111740228522727*x^(1)*y^(0) + 3.3508663077608247e-07*x^(
LinearRegression()
Mean Squared Error (MSE):     : 16736.027178290216
Root Mean Squared Error (RMSE): 129.3677980731303
R-squared (R2)                : 0.7897533203790228
```

```
#Predictions for linear regression

def predict_actual_values(model, poly_features, x1, x2):
    # Create polynomial features for the input values
    input_values = [[x1, x2]]
    input_poly = poly_features.transform(input_values)

    # Predict using the trained model
    predicted_value = model.predict(input_poly).flatten()

    return predicted_value


# Get user input for X1 and X2
x1_value = float(input("Enter the value for X1: "))
x2_value = float(input("Enter the value for X2: "))

predicted_result = predict_actual_values(model, poly_features, x1_value, x2_value)
actual_y = data.loc[(data['Density of Fluid (kg/m3)'] == x1_value) & (data['Bulk Modulus of Fluid (Pa)'] == x2_value),


print("Predicted Result:", predicted_result)
print(f"Actual Y from the dataset: {actual_y:.4f}")
```

```
Enter the value for X1: 1.57e3
Enter the value for X2: 2.21e9
Predicted Result: [1161.5574021]
Actual Y from the dataset: 1111.1100
```

```
# plot for linear regression degree 1

# Create a meshgrid for 3D plot
x1_col_range = np.linspace(x1_col.min(), x1_col.max(), 100)
x2_col_range = np.linspace(x2_col.min(), x2_col.max(), 100)
x1_col_mesh, x2_col_mesh = np.meshgrid(x1_col_range, x2_col_range)

# Combine the meshgrid features into a single array
meshgrid_features = np.c_[x1_col_mesh.flatten(), x2_col_mesh.flatten()]

# Create polynomial features for the meshgrid
meshgrid_poly = poly_features.transform(meshgrid_features)

# Predict the target variable for the meshgrid
y_col_mesh = model.predict(meshgrid_poly)
y_col_mesh = y_col_mesh.reshape(x1_col_mesh.shape)


# Scatter plot of the original data points
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x1_col, x2_col, y_col, color='blue', label='Original Data Points')

# Plot the fitted surface
ax.plot_surface(x1_col_mesh, x2_col_mesh, y_col_mesh, alpha=0.5, cmap='spring')

# Set axis labels
ax.set_xlabel('Density of Fluid (kg/m3)')
ax.set_ylabel('Bulk Modulus of Fluid (Pa)')
ax.set_zlabel('Sound wave speed in fluid (m/s)')

# Set the title
ax.set_title('Polynomial Regression Surface degree 1')

# Add a legend
ax.legend()


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



```
# for quadratic regression, degree 2

from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

x1_col = data['Density of Fluid (kg/m3)'].values
x2_col = data['Bulk Modulus of Fluid (Pa)'].values
y_col = data['Sound wave speed in fluid (m/s)'].values

# # Reshape the data to have a single feature
x1_col = x1_col.reshape(-1, 1)
x2_col = x2_col.reshape(-1, 1)
y_col = y_col.reshape(-1, 1)

# Combine the features into a single array
features = np.concatenate((x1_col, x2_col), axis=1)

# Create polynomial features
poly_features = PolynomialFeatures(degree=2)
features_poly = poly_features.fit_transform(features)

# Create and fit the linear regression model
model = LinearRegression()
model.fit(features_poly, y_col)

# Print the equation
equation = f"z = {model.intercept_[0]}"

for idx, coef in enumerate(model.coef_[0]):
    equation += f" + {coef}*x^({poly_features.powers_[idx][0]})*y^({poly_features.powers_[idx][1]})"

print("Equation:", equation)

# Predict the target variable
predicted_y_col = model.predict(features_poly)
print(model)
# Calculate evaluation metrics
mse = mean_squared_error(y_col, predicted_y_col)
rmse = np.sqrt(mse)
r2 = r2_score(y_col, predicted_y_col)

print("Mean Squared Error (MSE):     :", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared (R2)                :", r2)
print()
```

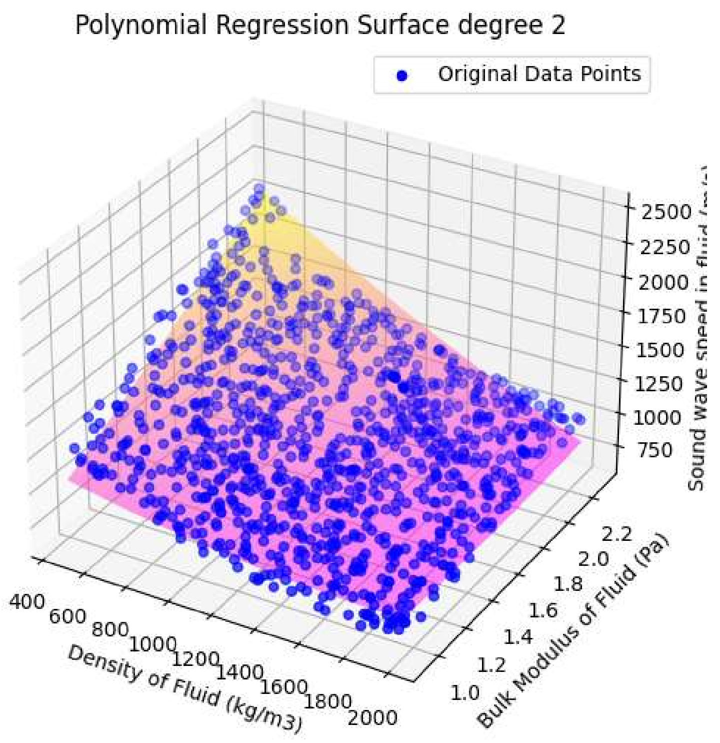
```
Equation: z = 505.44771562316 + 0.0*x^(0)*y^(0) + 3.0802747904453863e-08*x^(1)*y^(0) + 7.547372312200253e-07*x^(0)
LinearRegression()
Mean Squared Error (MSE):      : 13317.377906707707
Root Mean Squared Error (RMSE): 115.40094413265304
```

```
     R-squared (R2)                : 0.8327001709357226
```

```
#Predictions for quadratic regression, degree 2

def predict_actual_values(model, poly_features, x1, x2):
    # Create polynomial features for the input values
    input_values = [[x1, x2]]
    input_poly = poly_features.transform(input_values)

    # Predict using the trained model
    predicted_value = model.predict(input_poly).flatten()

    return predicted_value


# Get user input for X1 and X2
x1_value = float(input("Enter the value for X1: "))
x2_value = float(input("Enter the value for X2: "))

predicted_result = predict_actual_values(model, poly_features, x1_value, x2_value)
actual_y = data.loc[(data['Density of Fluid (kg/m3)'] == x1_value) & (data['Bulk Modulus of Fluid (Pa)'] == x2_value),


print("Predicted Result:", predicted_result)
print(f"Actual Y from the dataset: {actual_y:.4f}")
```

```
     Enter the value for X1: 1.57e3
     Enter the value for X2: 2.21e9
     Predicted Result: [1085.29133576]
     Actual Y from the dataset: 1111.1100
```

```
# plot for quadratic regression, degree 2

# Create a meshgrid for 3D plot
x1_col_range = np.linspace(x1_col.min(), x1_col.max(), 100)
x2_col_range = np.linspace(x2_col.min(), x2_col.max(), 100)
x1_col_mesh, x2_col_mesh = np.meshgrid(x1_col_range, x2_col_range)

# Combine the meshgrid features into a single array
meshgrid_features = np.c_[x1_col_mesh.flatten(), x2_col_mesh.flatten()]

# Create polynomial features for the meshgrid
meshgrid_poly = poly_features.transform(meshgrid_features)

# Predict the target variable for the meshgrid
y_col_mesh = model.predict(meshgrid_poly)
y_col_mesh = y_col_mesh.reshape(x1_col_mesh.shape)


# Scatter plot of the original data points
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x1_col, x2_col, y_col, color='blue', label='Original Data Points')

# Plot the fitted surface
ax.plot_surface(x1_col_mesh, x2_col_mesh, y_col_mesh, alpha=0.5, cmap='spring')

# Set axis labels
ax.set_xlabel('Density of Fluid (kg/m3)')
ax.set_ylabel('Bulk Modulus of Fluid (Pa)')
ax.set_zlabel('Sound wave speed in fluid (m/s)')

# Set the title
ax.set_title('Polynomial Regression Surface degree 2')

# Add a legend
ax.legend()


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


Polynomial Regression Surface degree 2
● Original Data Points
Density of Fluid (kg/m3)
400
600
800
1000
1200
1400
1600
1800
2000
Bulk Modulus of Fluid (Pa)
1.0
1.2
1.4
1.6
1.8
2.0
2.2
Sound wave speed in fluid (m/s)
750
1000
1250
1500
1750
2000
2250
2500

```
# for cubic regression, degree 3

from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

x1_col = data['Density of Fluid (kg/m3)'].values
x2_col = data['Bulk Modulus of Fluid (Pa)'].values
y_col = data['Sound wave speed in fluid (m/s)'].values

# # Reshape the data to have a single feature
x1_col = x1_col.reshape(-1, 1)
x2_col = x2_col.reshape(-1, 1)
y_col = y_col.reshape(-1, 1)

# Combine the features into a single array
features = np.concatenate((x1_col, x2_col), axis=1)

# Create polynomial features
poly_features = PolynomialFeatures(degree=3)
features_poly = poly_features.fit_transform(features)

# Create and fit the linear regression model
model = LinearRegression()
model.fit(features_poly, y_col)

# Print the equation
equation = f"z = {model.intercept_[0]}"

for idx, coef in enumerate(model.coef_[0]):
    equation += f" + {coef}*x^({poly_features.powers_[idx][0]})*y^({poly_features.powers_[idx][1]})"

print("Equation:", equation)

# Predict the target variable
predicted_y_col = model.predict(features_poly)
print(model)
# Calculate evaluation metrics
mse = mean_squared_error(y_col, predicted_y_col)
rmse = np.sqrt(mse)
r2 = r2_score(y_col, predicted_y_col)

print("Mean Squared Error (MSE):     :", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared (R2)                :", r2)
print()
```

```
Equation: z = 771.5626915429623 + 0.0*x^(0)*y^(0) + -5.79131243017189e-20*x^(1)*y^(0) + -4.691355841747578e-24*x^(
LinearRegression()
Mean Squared Error (MSE):     : 17316.897853876068
Root Mean Squared Error (RMSE): 131.5936847036212
R-squared (R2)                : 0.7824561207790163
```

```
#Predictions for cubic regression, degree 3

def predict_actual_values(model, poly_features, x1, x2):
    # Create polynomial features for the input values
    input_values = [[x1, x2]]
    input_poly = poly_features.transform(input_values)

    # Predict using the trained model
    predicted_value = model.predict(input_poly).flatten()
```

```
# plot for cubic regression, degree 3

# Create a meshgrid for 3D plot
x1_col_range = np.linspace(x1_col.min(), x1_col.max(), 100)
x2_col_range = np.linspace(x2_col.min(), x2_col.max(), 100)
x1_col_mesh, x2_col_mesh = np.meshgrid(x1_col_range, x2_col_range)

# Combine the meshgrid features into a single array
meshgrid_features = np.c_[x1_col_mesh.flatten(), x2_col_mesh.flatten()]

# Create polynomial features for the meshgrid
meshgrid_poly = poly_features.transform(meshgrid_features)

# Predict the target variable for the meshgrid
y_col_mesh = model.predict(meshgrid_poly)
y_col_mesh = y_col_mesh.reshape(x1_col_mesh.shape)


# Scatter plot of the original data points
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x1_col, x2_col, y_col, color='blue', label='Original Data Points')
```