

```

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error

data = pd.read_csv('/content/displacement_data.csv')
dfx= data[["x","y","u_x"]]
dfy= data[["x","y","u_y"]]

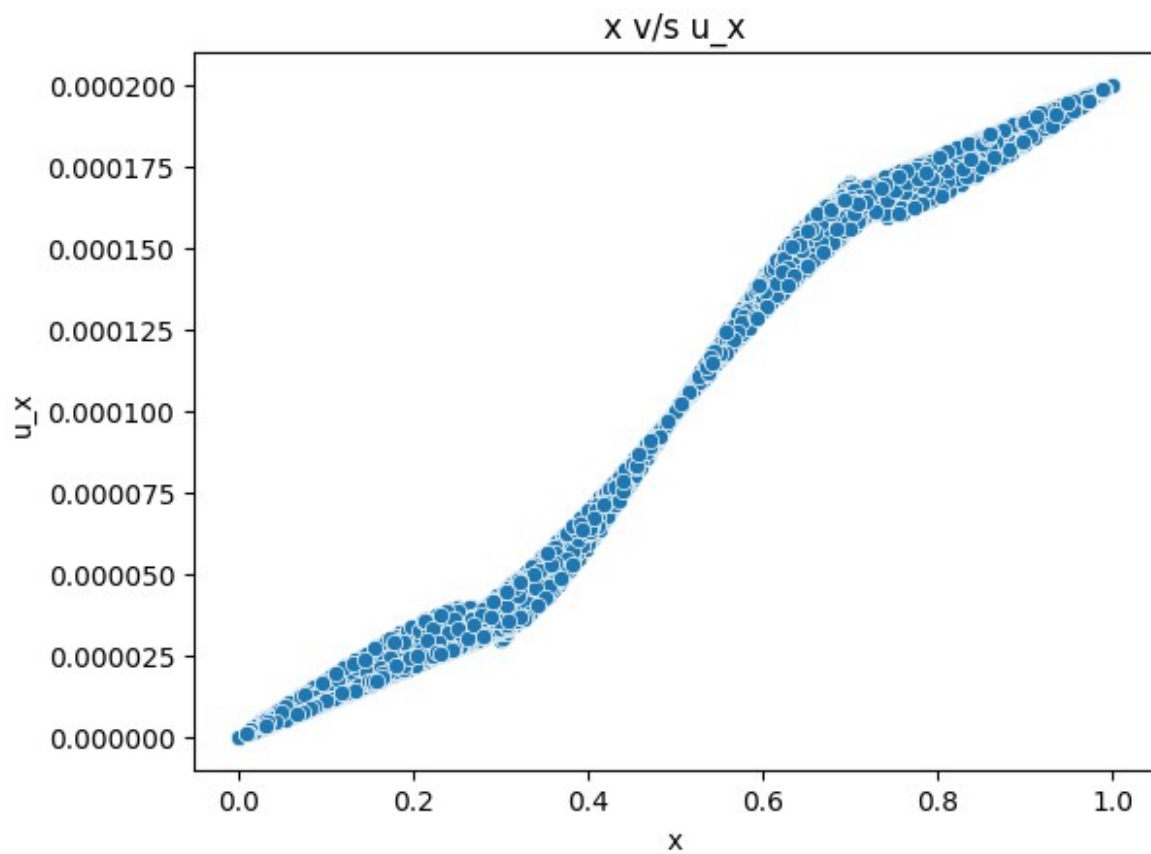
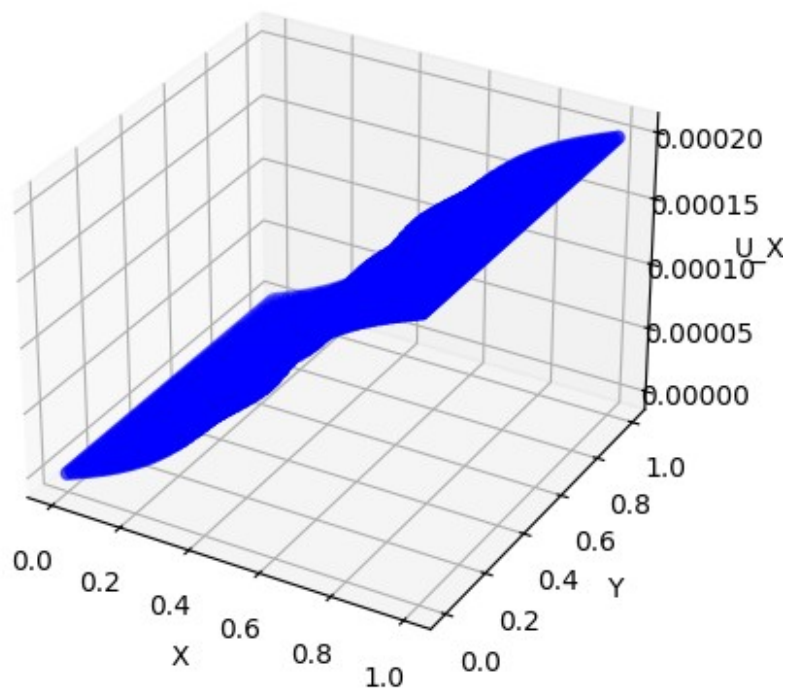
x1 = dfx[['x', 'y']]
y1 = dfx['u_x']
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(dfx['x'], dfx['y'], dfx['u_x'], c='b', marker='o')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('U_X')
plt.title("3D Scatter Plot of X, Y and UX")
plt.show()

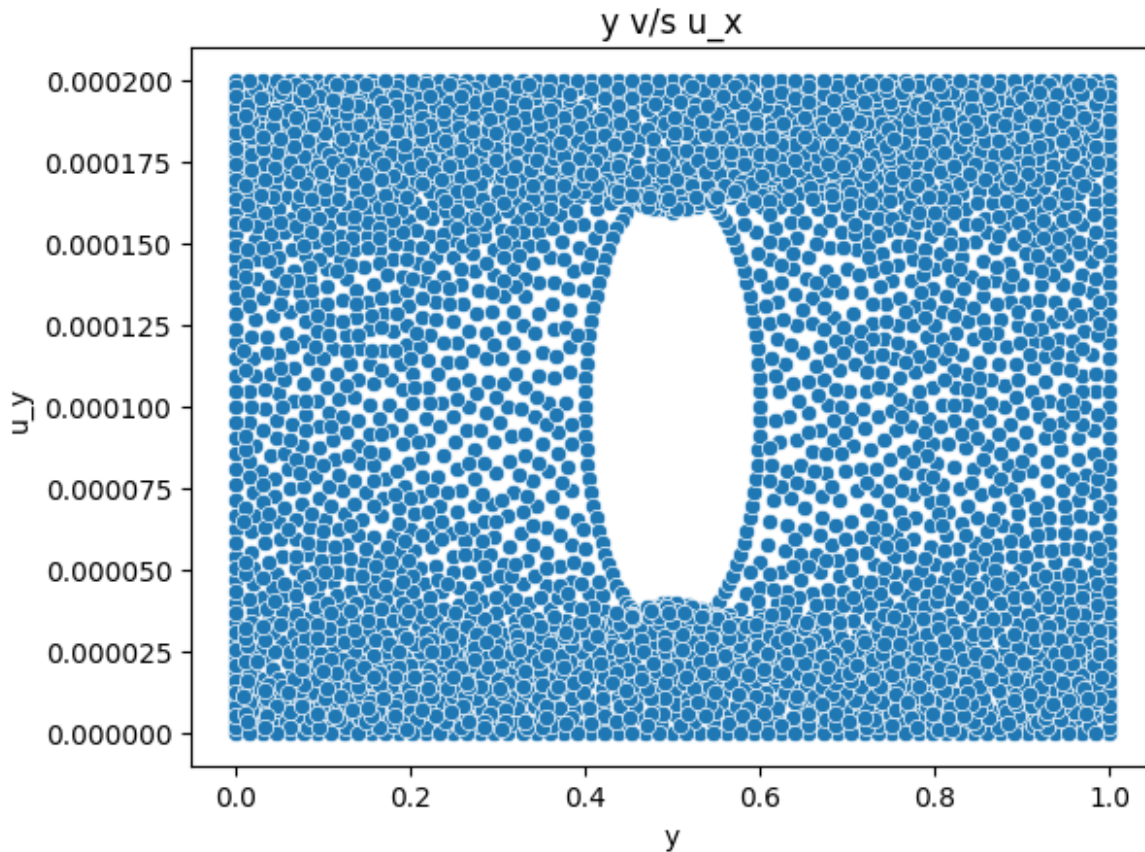
sns.scatterplot(x="x",y="u_x",data=dfx)
plt.title("x v/s u_x")
plt.xlabel("x")
plt.ylabel("u_x")
plt.show()

sns.scatterplot(x="y",y="u_x",data=dfx)
plt.title("y v/s u_x")
plt.xlabel("y")
plt.ylabel("u_y")
plt.show()

```

3D Scatter Plot of X, Y and UX





```

degree = 3
poly = PolynomialFeatures(degree)
X_poly = poly.fit_transform(x1)
model_x = LinearRegression()
model_x.fit(X_poly,y1)
x_range = np.linspace(dfx['x'].min(), dfx['x'].max(), 100)
y_range = np.linspace(dfx['y'].min(), dfx['y'].max(), 100)
X_grid, Y_grid = np.meshgrid(x_range, y_range)
X_grid_poly = poly.transform(np.c_[X_grid.ravel(), Y_grid.ravel()])
Z_grid = model_x.predict(X_grid_poly).reshape(X_grid.shape)

# Plot the original data and the polynomial plane
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

# Plot the original scatter data
ax.scatter(dfx['x'], dfx['y'], dfx['u_x'], color='blue', alpha=0.5,
label="Original Data")

# Plot the polynomial plane
ax.plot_surface(X_grid, Y_grid, Z_grid, color='orange', alpha=0.7,
rstride=100, cstride=100, edgecolor='k')
ax.set_xlabel('X')

```

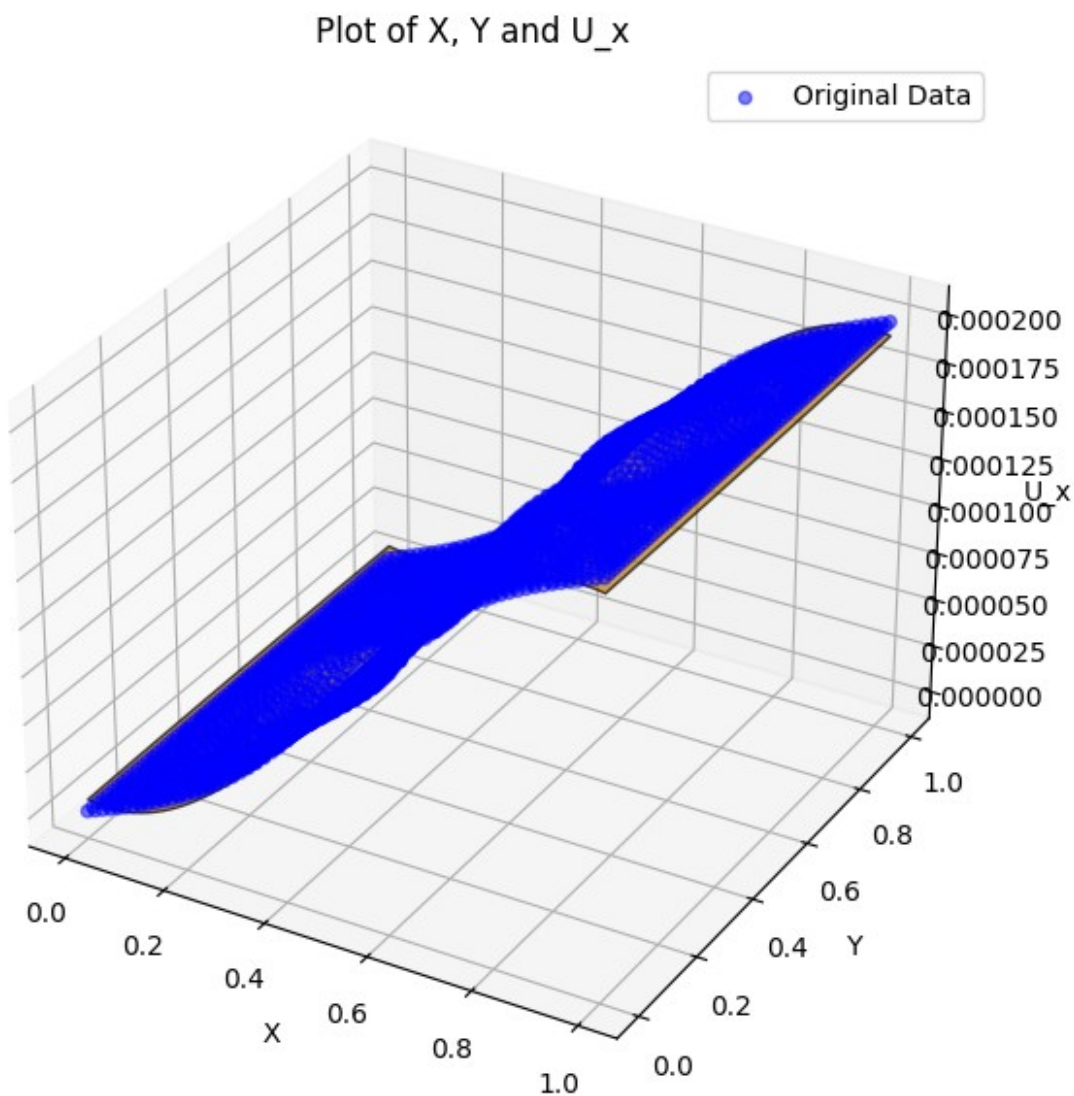
```

ax.set_ylabel('Y')
ax.set_zlabel('U_x')
ax.set_title('Plot of X, Y and U_x')

plt.legend()
plt.show()
u_x_pred = model_x.predict(X_poly)
print("Mean Squared Error:", mean_squared_error(y1, u_x_pred))

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:493:
UserWarning: X does not have valid feature names, but
PolynomialFeatures was fitted with feature names
  warnings.warn(

```



Mean Squared Error: 2.7972761440991277e-11

```

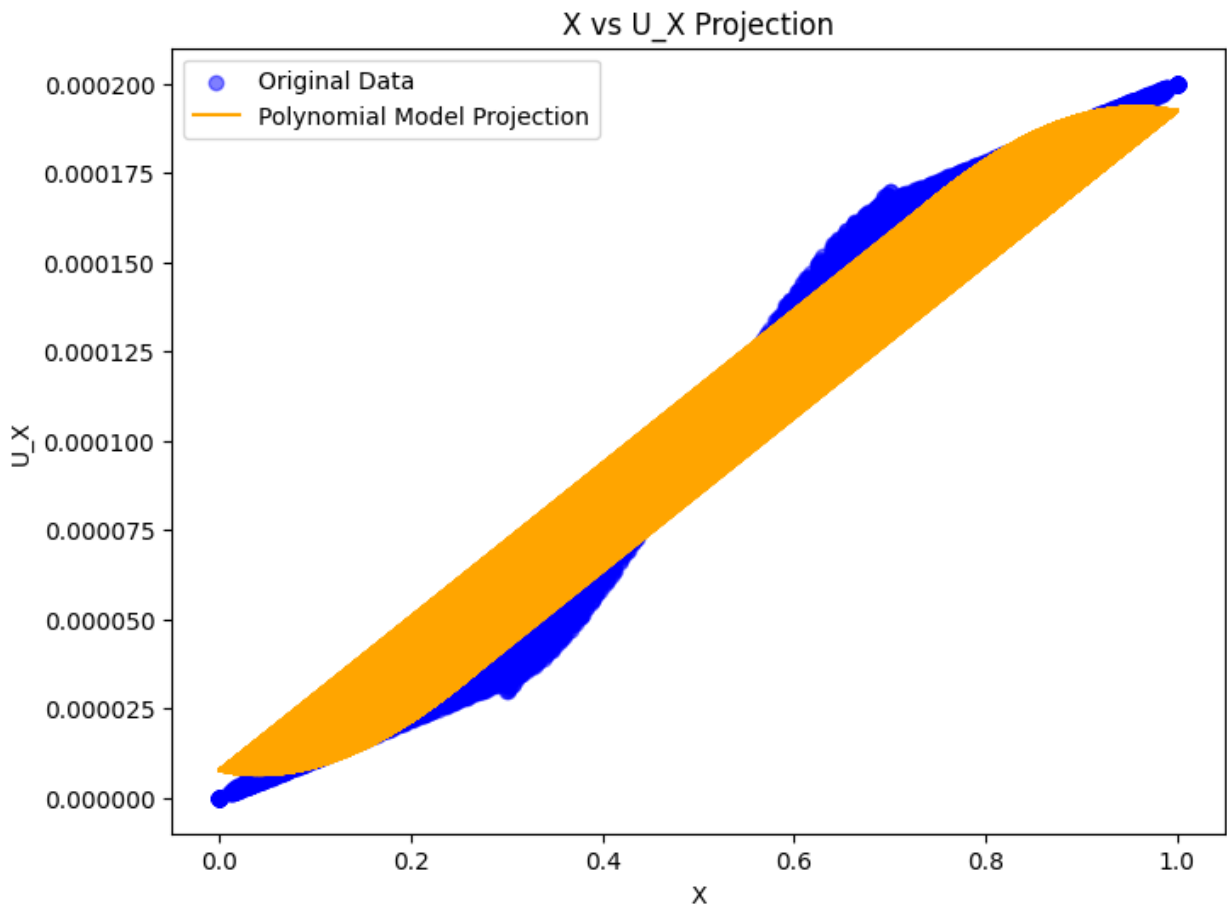
# Define a fixed value for y (e.g., the mean of y)
y_fixed = np.full_like(dfx['x'], dfx['y'].mean())

# Create a DataFrame with varying x and fixed y
X_x_projection = pd.DataFrame({'x': dfx['x'], 'y': y_fixed})
X_x_projection_poly = poly.transform(X_x_projection) # Transform for
polynomial terms

# Predict u_y along x-axis using the fixed y value
u_y_x_projection = model_x.predict(X_x_projection_poly)

# Plot the original data projection along the x-axis
plt.figure(figsize=(8, 6))
plt.scatter(dfx['x'], dfx['u_x'], color='blue', alpha=0.5,
label='Original Data')
plt.plot(dfx['x'], u_y_x_projection, color='orange', label='Polynomial
Model Projection')
plt.xlabel('X')
plt.ylabel('U_X')
plt.title('X vs U_X Projection')
plt.legend()
plt.show()

```



```

# Extract terms and construct the polynomial equation string
coefficients_x = model_x.coef_
print(coefficients_x)
intercept_x = model_x.intercept_
print(intercept_x)
terms = poly.get_feature_names_out(['x', 'y'])
equation = f"{intercept_x} " # Start with the intercept

for coef, term in zip(coefficients_x[1:], terms[1:]): # Skip the
first term as it corresponds to the intercept
    equation += f"+ ({coef}) * {term} "

print("Polynomial Regression Equation for u_x:")
print(equation)

[ 0.00000000e+00 -5.47907372e-05  3.62073969e-06  7.24236347e-04
 -7.06826373e-06 -4.24450232e-06 -4.83291020e-04  1.29737850e-06
  5.58055387e-06  1.14017395e-06]
6.768432748859872e-06
Polynomial Regression Equation for u_x:
6.768432748859872e-06 + (-5.4790737241767054e-05) * x +
(3.620739685636638e-06) * y + (0.0007242363471388798) * x^2 + (-
7.06826373418302e-06) * x y + (-4.244502323836039e-06) * y^2 + (-
0.0004832910195154034) * x^3 + (1.297378498170553e-06) * x^2 y +
(5.580553867172199e-06) * x y^2 + (1.140173949064537e-06) * y^3

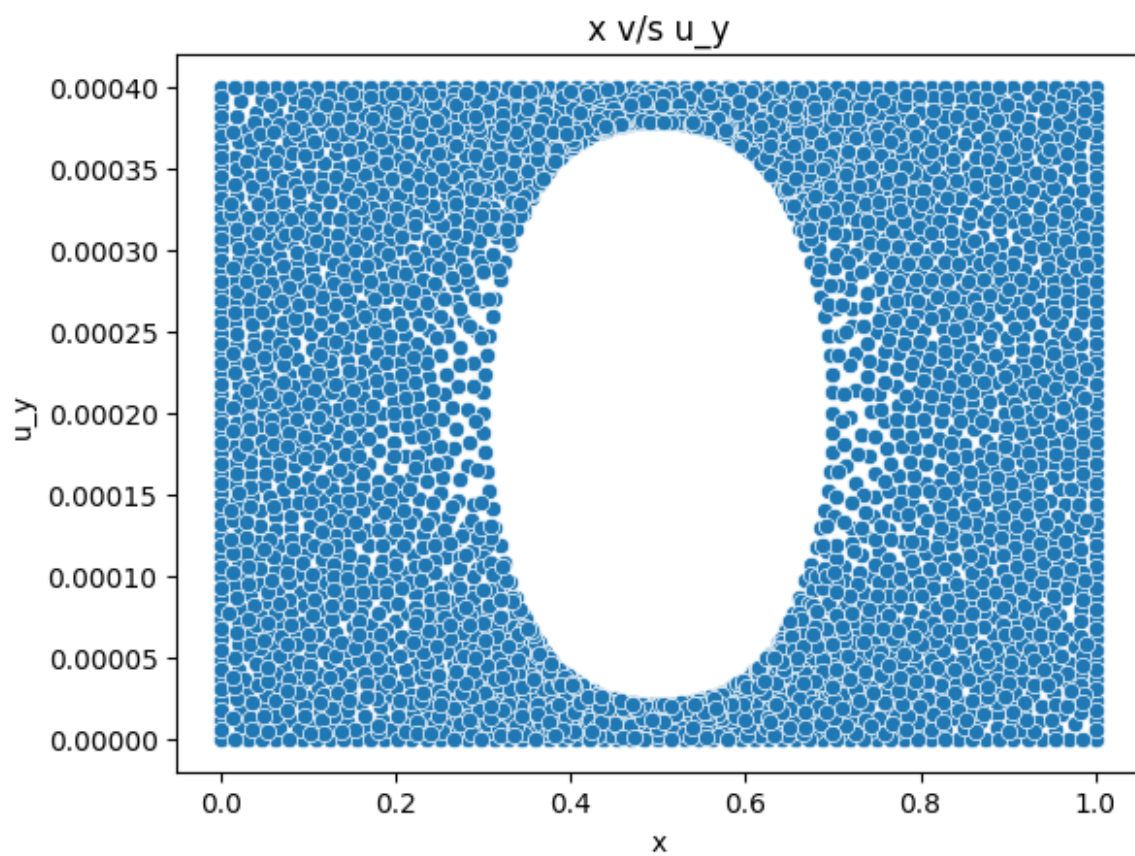
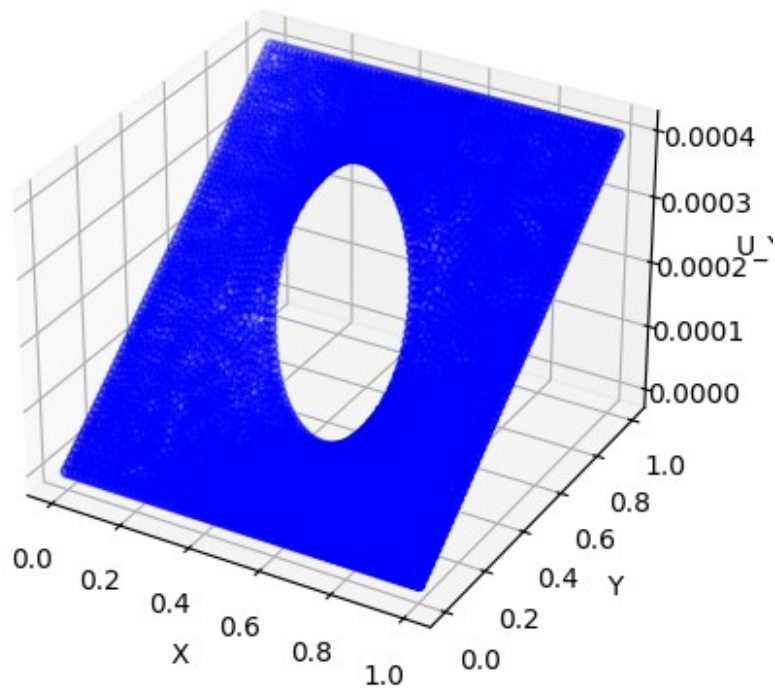
x2 = dfy[['x', 'y']]
y2 = dfy['u_y']
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(dfy['x'], dfy['y'], dfy['u_y'], c='b', marker='o')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('U_Y')
plt.title("3D Scatter Plot of X, Y and UY")
plt.show()

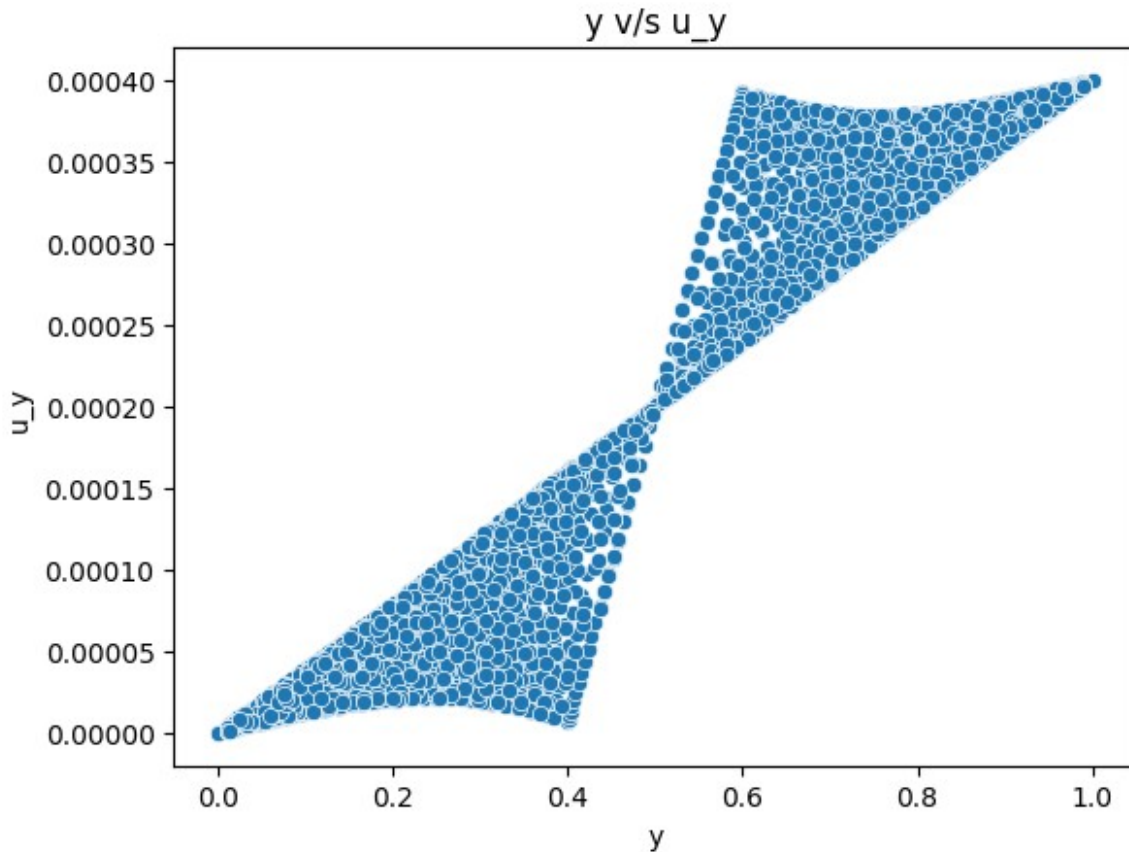
sns.scatterplot(x="x",y="u_y",data=dfy)
plt.title("x v/s u_y")
plt.xlabel("x")
plt.ylabel("u_y")
plt.show()

sns.scatterplot(x="y",y="u_y",data=dfy)
plt.title("y v/s u_y")
plt.xlabel("y")
plt.ylabel("u_y")
plt.show()

```


3D Scatter Plot of X, Y and UY





```

degree = 3
poly = PolynomialFeatures(degree)
X_poly = poly.fit_transform(x2)

# Fit a linear regression model on the transformed features
model_y = LinearRegression()
model_y.fit(X_poly, y2)

# Predict u_y values for original X
u_y_pred = model_y.predict(X_poly)

# Print model coefficients
print("Polynomial Coefficients:", model_y.coef_)
print("Intercept:", model_y.intercept_)
print("Mean Squared Error:", mean_squared_error(y2, u_y_pred))
coefficients_y = model_y.coef_
intercept_y = model_y.intercept_
terms = poly.get_feature_names_out(['x', 'y'])
equation = f"{intercept_y} " # Start with the intercept

for coef, term in zip(coefficients_y[1:], terms[1:]): # Skip the
first term as it corresponds to the intercept
    equation += f"+ ({coef}) * {term} "

```



```

print("Polynomial Regression Equation for u_y:")
print(equation)

Polynomial Coefficients: [ 0.00000000e+00 -2.96290761e-04 -
2.17928468e-04  2.95686718e-04
 5.97648050e-04  1.45769144e-03  2.46549477e-06 -6.02551062e-04
 5.03792662e-06 -9.73136590e-04]
Intercept: 6.561611075398785e-05
Mean Squared Error: 8.496373380888251e-10
Polynomial Regression Equation for u_y:
6.561611075398785e-05 + (-0.00029629076140344317) * x + (-
0.0002179284680354217) * y + (0.00029568671766650335) * x^2 +
(0.0005976480500583075) * x y + (0.001457691440835397) * y^2 +
(2.465494766317093e-06) * x^3 + (-0.0006025510620320173) * x^2 y +
(5.037926617474617e-06) * x y^2 + (-0.0009731365903089235) * y^3

x_range = np.linspace(dfy['x'].min(), dfy['x'].max(), 100)
y_range = np.linspace(dfy['y'].min(), dfy['y'].max(), 100)
X_grid, Y_grid = np.meshgrid(x_range, y_range)
X_grid_poly = poly.transform(np.c_[X_grid.ravel(), Y_grid.ravel()])
Z_grid = model_y.predict(X_grid_poly).reshape(X_grid.shape)

# Plot the original data and the polynomial plane
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

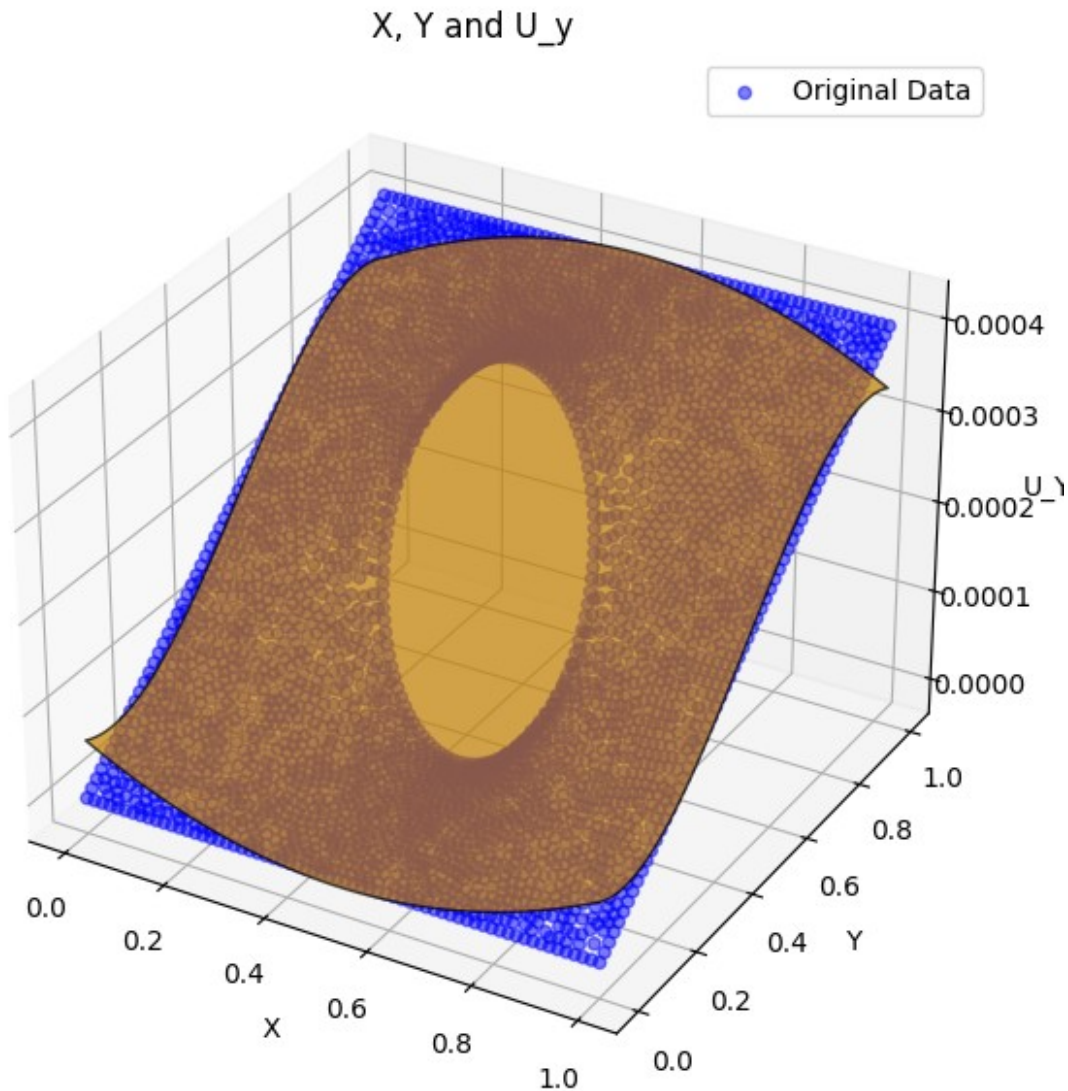
# Plot the original scatter data
ax.scatter(dfy['x'], dfy['y'], dfy['u_y'], color='blue', alpha=0.5,
label="Original Data")

# Plot the polynomial plane
ax.plot_surface(X_grid, Y_grid, Z_grid, color='orange', alpha=0.7,
rstride=100, cstride=100, edgecolor='k')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('U_Y')
ax.set_title('X, Y and U_y')

plt.legend()
plt.show()

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:493:
UserWarning: X does not have valid feature names, but
PolynomialFeatures was fitted with feature names
warnings.warn(

```



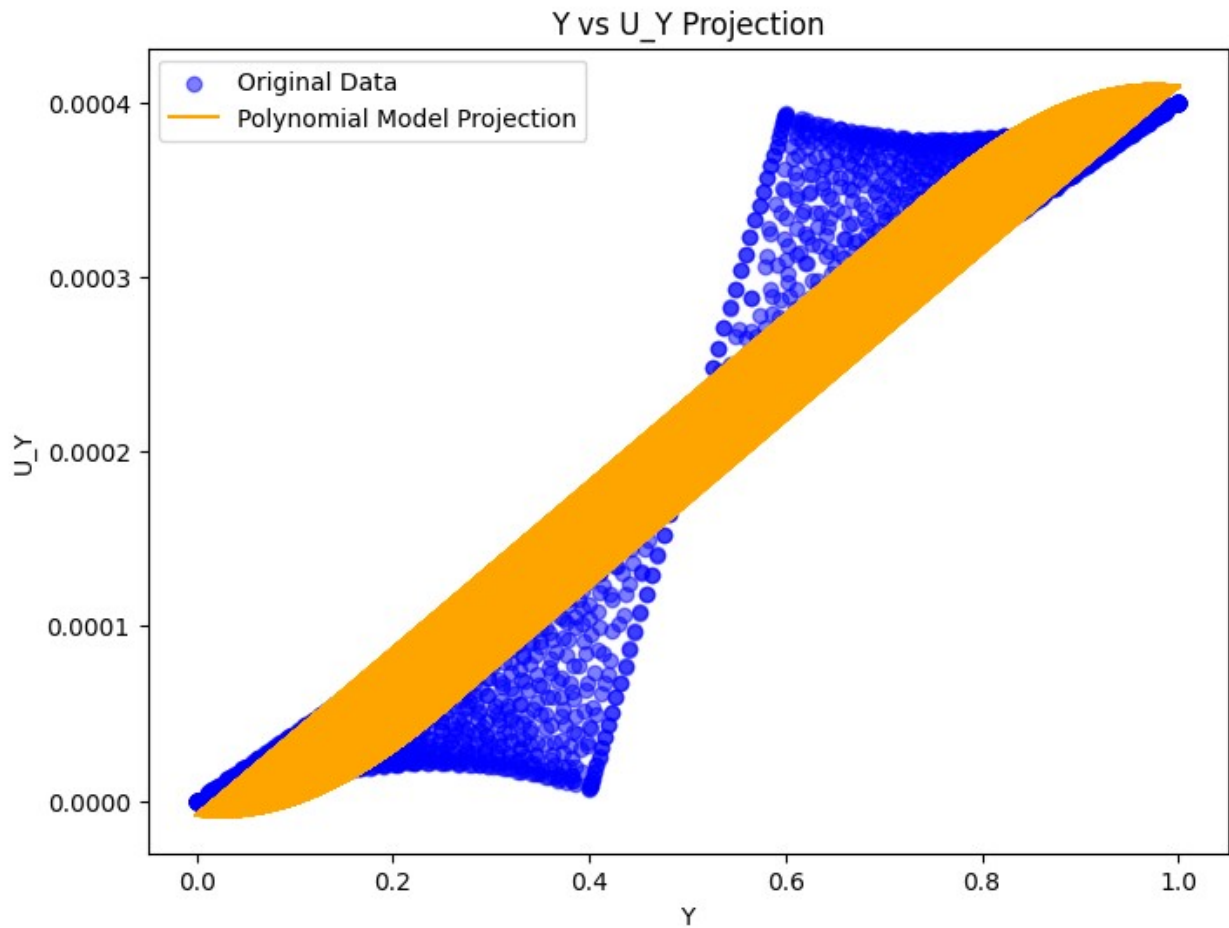
```
x_fixed = np.full_like(dfy['y'], dfy['x'].mean())

# Create a DataFrame with fixed x and varying y
X_y_projection = pd.DataFrame({'x': x_fixed, 'y': dfy['y']})
X_y_projection_poly = poly.transform(X_y_projection) # Transform for
polynomial terms

# Predict u_y along y-axis using the fixed x value
u_y_y_projection = model_y.predict(X_y_projection_poly)

# Plot the original data projection along the y-axis
plt.figure(figsize=(8, 6))
plt.scatter(dfy['y'], dfy['u_y'], color='blue', alpha=0.5,
label='Original Data')
plt.plot(dfy['y'], u_y_y_projection, color='orange', label='Polynomial
Model Projection')
```

```
plt.xlabel('Y')
plt.ylabel('U_Y')
plt.title('Y vs U_Y Projection')
plt.legend()
plt.show()
```



```
from sympy import symbols, diff
x,y = symbols('x y')
ux=(
    intercept_x+
    coefficients_x[1]*x+
    coefficients_x[2]*y+
    coefficients_x[3]*x**2+
    coefficients_x[4]*x*y+
    coefficients_x[5]*y**2+
    coefficients_x[6]*x**3+
    coefficients_x[7]*x**2*y+
    coefficients_x[8]*x*y**2+
    coefficients_x[9]*y**3
)
```

```

)
print(ux)

-0.000483291019515403*x**3 + 1.29737849817055e-6*x**2*y +
0.00072423634713888*x**2 + 5.5805538671722e-6*x*y**2 -
7.06826373418302e-6*x*y - 5.47907372417671e-5*x + 1.14017394906454e-
6*y**3 - 4.24450232383604e-6*y**2 + 3.62073968563664e-6*y +
6.76843274885987e-6

uy=(
    intercept_y+
    coefficients_y[1]*x+
    coefficients_y[2]*y+
    coefficients_y[3]*x**2+
    coefficients_y[4]*x*y+
    coefficients_y[5]*y**2+
    coefficients_y[6]*x**3+
    coefficients_y[7]*x**2*y+
    coefficients_y[8]*x*y**2+
    coefficients_y[9]*y**3
)
print(uy)

2.46549476631709e-6*x**3 - 0.000602551062032017*x**2*y +
0.000295686717666503*x**2 + 5.03792661747462e-6*x*y**2 +
0.000597648050058308*x*y - 0.000296290761403443*x -
0.000973136590308923*y**3 + 0.0014576914408354*y**2 -
0.000217928468035422*y + 6.56161107539879e-5

exx_1=diff(ux,x)
eyy_1=diff(uy,y)
exx = sum(term for term in exx_1.as_ordered_terms() if term.has(x, y))
eyy = sum(term for term in eyy_1.as_ordered_terms() if term.has(x, y))
print(exx)
print(eyy)

-0.00144987305854621*x**2 + 2.59475699634111e-6*x*y +
0.00144847269427776*x + 5.5805538671722e-6*y**2 - 7.06826373418302e-
6*y
-0.000602551062032017*x**2 + 1.00758532349492e-5*x*y +
0.000597648050058308*x - 0.00291940977092677*y**2 +
0.00291538288167079*y

s_xx_lambda= sum([exx,eyy])
s_yy_lambda= s_xx_lambda
s_xx_mew= 2*exx
s_yy_mew= 2*eyy

print(s_xx_lambda)
print(s_yy_lambda)

```

```

print(s_xx_mew)
print(s_yy_mew)

-0.00205242412057823*x**2 + 1.26706102312903e-5*x*y +
0.00204612074433607*x - 0.0029138292170596*y**2 +
0.00290831461793661*y
-0.00205242412057823*x**2 + 1.26706102312903e-5*x*y +
0.00204612074433607*x - 0.0029138292170596*y**2 +
0.00290831461793661*y
-0.00289974611709242*x**2 + 5.18951399268221e-6*x*y +
0.00289694538855552*x + 1.11611077343444e-5*y**2 - 1.4136527468366e-
5*y
-0.00120510212406403*x**2 + 2.01517064698985e-5*x*y +
0.00119529610011662*x - 0.00583881954185354*y**2 +
0.00583076576334159*y

R_x_1=s_xx_lambda.subs(x,1)
R_x_2=s_xx_mew.subs(x,1)
print(R_x_1,R_x_2)

-0.0029138292170596*y**2 + 0.0029209852281679*y - 6.30337624216073e-6
1.11611077343444e-5*y**2 - 8.94701347568383e-6*y - 2.80072853690123e-6

from sympy import integrate
R_x_lambda=integrate(R_x_1, (y, 0, 1))
R_x_mew=integrate(R_x_2, (y, 0, 1))
print(R_x_lambda)
print(R_x_mew)

0.000482912832155257
-3.55386602996168e-6

R_y_1=s_yy_lambda.subs(y,1)
R_y_2=s_yy_mew.subs(y,1)
R_y_lambda=integrate(R_y_1, (x, 0, 1))
R_y_mew=integrate(R_y_2, (x, 0, 1))
print(R_y_lambda)
print(R_y_mew)

0.000339739704634615
0.000197969416759959

Reaction = pd.read_csv('/content/reaction_data.csv')
Reaction.head()

{"summary":{"\n  \"name\": \"Reaction\", \n  \"rows\": 4, \n
\"fields\": [\n    {\n      \"column\": \"Reaction\", \n
\"properties\": {\n        \"dtype\": \"string\", \n
\"num_unique_values\": 4, \n        \"samples\": [\n          \"R4\", \n
\"R2\", \n          \"R3\", \n        ], \n        \"semantic_type\":
\"\", \n        \"description\": \"\" \n      } \n    }, \n    {\n

```

```
R_x=Reaction.iloc[1,1]
R_y=Reaction.iloc[3,1]
C=np.array([R_x,R_y],dtype = float)
print(C)
```

```
A=np.array([[R_x_lambda,R_x_mew],[R_y_lambda,R_y_mew]],dtype = float)
print(A)
```

```
l= np.linalg.solve(A, C)
print(f"Lame's constants are:- {l}")
```

```
s_xx= l[0]*s_xx_lambda+(l[1]*s_xx_mew)
s_yy= l[0]*s_yy_lambda+(l[1]*s_yy_mew)
from sympy import lambdify
s_x= lambdify((x,y),s_xx)
s_y = lambdify((x,y),s_yy)
e_x= lambdify((x,y),exx)
e_y = lambdify((x,y),eyy)
data['stress_x'] = np.vectorize(s_x)(data['x'], data['y'])
data['stress_y'] = np.vectorize(s_y)(data['x'], data['y'])
data['e_x'] = np.vectorize(e_x)(data['x'], data['y'])
data['e_y'] = np.vectorize(e_y)(data['x'], data['y'])
```

```

{"summary": "{\n  \"name\": \"data\", \n  \"rows\": 3877, \n  \"fields\": [\n    {\n      \"column\": \"x\", \n      \"properties\": {\n        \"dtype\": \"number\", \n        \"std\": 0.29956252910327485, \n        \"min\": 0.0, \n        \"max\": 1.0, \n        \"num_unique_values\": 3623, \n        \"samples\": [\n          0.0508864939110139, \n          0.1261502602372457, \n          0.2118536552724418\n        ], \n        \"semantic_type\": \"\", \n        \"description\": \"\" }\n    }, \n    {\n      \"column\": \"y\", \n      \"properties\": {\n        \"dtype\": \"number\", \n        \"std\": 0.2990539284472939, \n        \"min\": 0.0, \n        \"max\": 1.0, \n        \"num unique values\": 

```



```

3599,\n          \"samples\": [\n          0.978615252350456,\n0.0629078544311327,\n          0.6518433923214456\n          ],\n\n\"semantic_type\": \"\", \n          \"description\": \"\" \n          }\n          },\n          {\n          \"column\": \"u_x\", \n          \"properties\": {\n          \"dtype\": \"number\", \n          \"std\": 7.013350042008814e-05, \n          \"min\": 0.0, \n          \"max\": 0.0002, \n          \"num_unique_values\": 3749, \n          \"samples\": [\n          0.0001519904113859, \n          0.0001663117663906, \n          0.0001567432501386\n          ], \n          \"semantic_type\": \"\", \n          \"description\": \"\" \n          }\n          },\n          {\n          \"column\": \"u_y\", \n          \"properties\": {\n          \"dtype\": \"number\", \n          \"std\": 0.00014219059621856526, \n          \"min\": 0.0, \n          \"max\": 0.0004, \n          \"num_unique_values\": 3749, \n          \"samples\": [\n          2.2892151648526507e-05, \n          9.968463481922928e-05, \n          5.165557971057465e-05\n          ], \n          \"semantic_type\": \"\", \n          \"description\": \"\" \n          }\n          },\n          {\n          \"column\": \"stress_x\", \n          \"properties\": {\n          \"dtype\": \"number\", \n          \"std\": 81631237.95195091, \n          \"min\": -1702705.5748360157, \n          \"max\": 364706762.39154196, \n          \"num_unique_values\": 3877, \n          \"samples\": [\n          260566124.06053782, \n          299574071.69539964, \n          289521713.12202823\n          ], \n          \"semantic_type\": \"\", \n          \"description\": \"\" \n          }\n          },\n          {\n          \"column\": \"stress_y\", \n          \"properties\": {\n          \"dtype\": \"number\", \n          \"std\": 137438821.75621462, \n          \"min\": -3153382.2616819143, \n          \"max\": 567290416.9765737, \n          \"num_unique_values\": 3877, \n          \"samples\": [\n          241195937.60099816, \n          538951776.6316009, \n          481378899.74264663\n          ], \n          \"semantic_type\": \"\", \n          \"description\": \"\" \n          }\n          },\n          {\n          \"column\": \"e_x\", \n          \"properties\": {\n          \"dtype\": \"number\", \n          \"std\": 0.00011076078153473858, \n          \"min\": -2.2967170022293795e-06, \n          \"max\": 0.0003617680825023275, \n          \"num_unique_values\": 3877, \n          \"samples\": [\n          0.0003490371624448099, \n          0.00025523050426935203, \n          0.0002687094763293669\n          ], \n          \"semantic_type\": \"\", \n          \"description\": \"\" \n          }\n          },\n          {\n          \"column\": \"e_y\", \n          \"properties\": {\n          \"dtype\": \"number\", \n          \"std\": 0.00022491044878129838, \n          \"min\": -4.903011973709029e-06, \n          \"max\": 0.000854469161381062, \n          \"num_unique_values\": 3877, \n          \"samples\": [\n          0.00030226799618730295, \n          0.0008332061321181593, \n          0.0007319471781191474\n          ], \n          \"semantic_type\": \"\", \n          \"description\": \"\" \n          }\n          }\n          ]\n          },\n          \"type\": \"dataframe\", \"variable_name\": \"data\"}

```

```

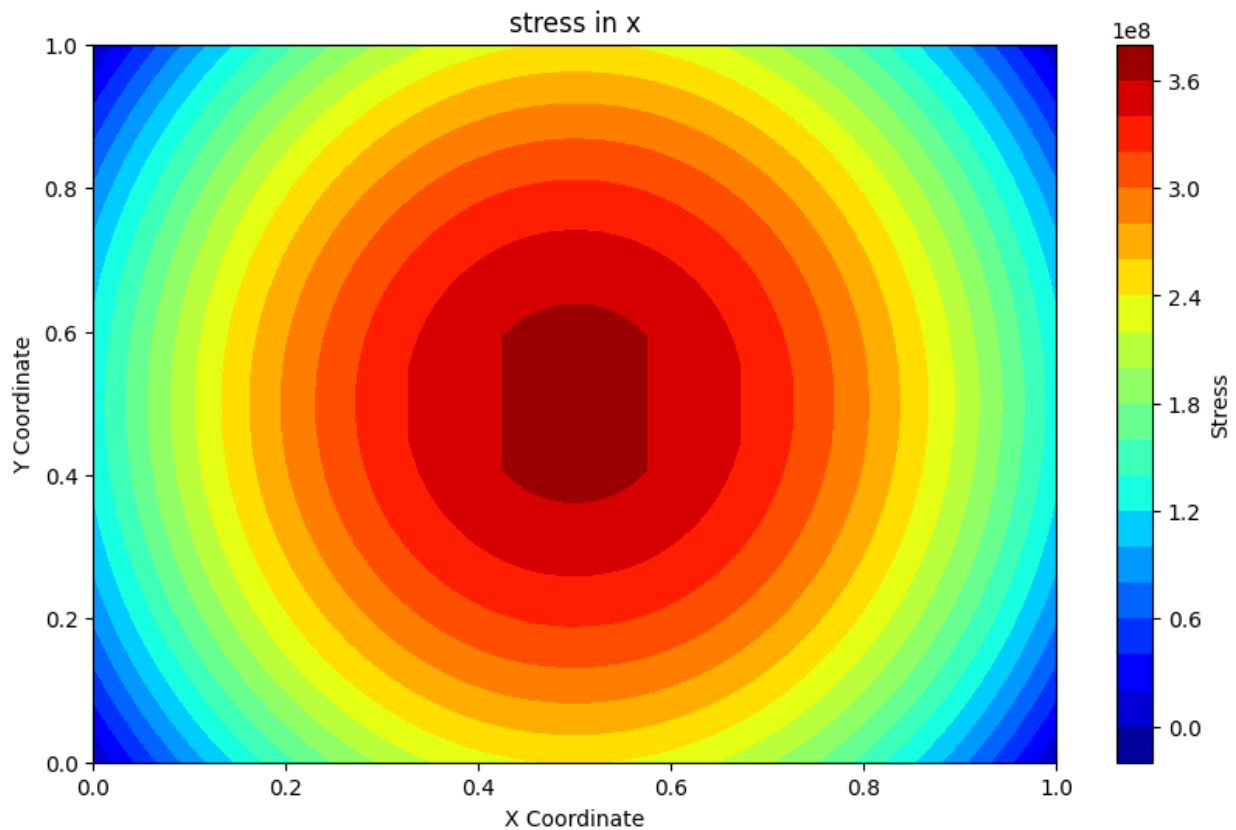
plt.figure(figsize=(10, 6))
contour = plt.tricontourf(data['x'], data['y'], data['stress_x'],
levels=20, cmap="jet")

```

```
# Step 2: Add a color bar for reference
cbar = plt.colorbar(contour)
cbar.set_label("Stress")

# Optional: Add labels, title, etc.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.title("stress in x")

plt.show()
```

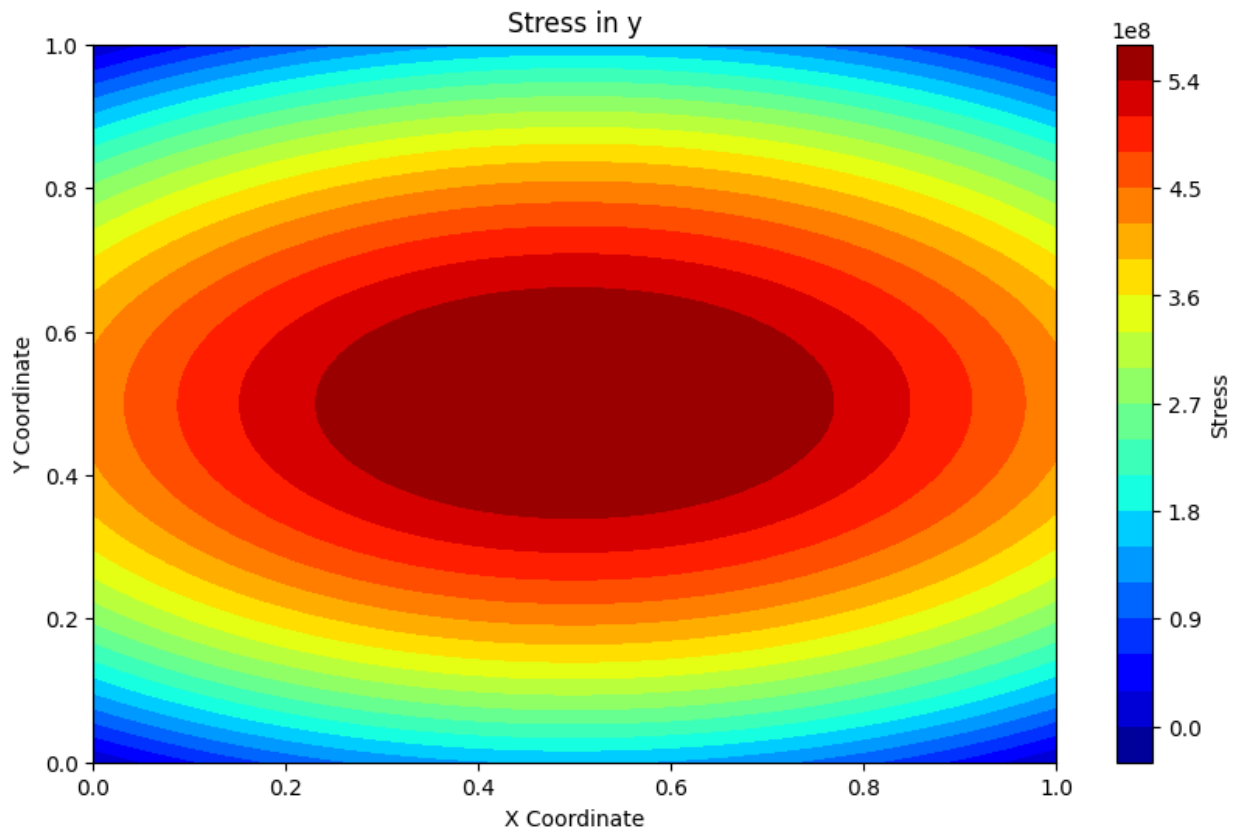


```
plt.figure(figsize=(10, 6))
contour = plt.tricontourf(data['x'], data['y'], data['stress_y'],
levels=20, cmap="jet")

# Step 2: Add a color bar for reference
cbar = plt.colorbar(contour)
cbar.set_label("Stress")

# Optional: Add labels, title, etc.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.title("Stress in y")
```

```
plt.show()
```



```
Txy=(l[1])*(diff(ux,y)+diff(uy,x))
T_xy=lambdify((x,y),Txy)
data['Txy'] = np.vectorize(T_xy)(data['x'], data['y'])
data.head()

{"summary":{"\n  \"name\": \"data\", \n  \"rows\": 3877, \n  \"fields\": [\n    {\n      \"column\": \"x\", \n      \"properties\": {\n        \"dtype\": \"number\", \n        \"std\": 0.29956252910327485, \n        \"min\": 0.0, \n        \"max\": 1.0, \n        \"num_unique_values\": 3623, \n        \"samples\": [\n          0.0508864939110139, \n          0.1261502602372457, \n          0.2118536552724418\n        ], \n        \"semantic_type\": \"\", \n        \"description\": \"\"\n      }\n    }, \n    {\n      \"column\": \"y\", \n      \"properties\": {\n        \"dtype\": \"number\", \n        \"std\": 0.2990539284472939, \n        \"min\": 0.0, \n        \"max\": 1.0, \n        \"num_unique_values\": 3599, \n        \"samples\": [\n          0.978615252350456, \n          0.0629078544311327, \n          0.6518433923214456\n        ], \n        \"semantic_type\": \"\", \n        \"description\": \"\"\n      }\n    }, \n    {\n      \"column\": \"u_x\", \n      \"properties\": {\n        \"dtype\": \"number\", \n        \"std\": 7.013350042008814e-05, \n
```

```

{"min": 0.0, "max": 0.0002, "num_unique_values": 3749, "samples": [0.0001519904113859, 0.0001663117663906, 0.0001567432501386], "semantic_type": "", "description": "", "column": "u_y", "properties": {"dtype": "number", "std": 0.00014219059621856526, "min": 0.0, "max": 0.0004, "num_unique_values": 3749, "samples": [2.2892151648526507e-05, 9.968463481922928e-05, 5.165557971057465e-05], "semantic_type": "", "description": "", "column": "stress_x", "properties": {"dtype": "number", "std": 81631237.95195091, "min": -1702705.5748360157, "max": 364706762.39154196, "num_unique_values": 3877, "samples": [260566124.06053782, 299574071.69539964, 289521713.12202823], "semantic_type": "", "description": "", "column": "stress_y", "properties": {"dtype": "number", "std": 137438821.75621462, "min": -3153382.2616819143, "max": 567290416.9765737, "num_unique_values": 3877, "samples": [241195937.60099816, 538951776.6316009, 481378899.74264663], "semantic_type": "", "description": "", "column": "e_x", "properties": {"dtype": "number", "std": 0.00011076078153473858, "min": -2.2967170022293795e-06, "max": 0.0003617680825023275, "num_unique_values": 3877, "samples": [0.0003490371624448099, 0.00025523050426935203, 0.0002687094763293669], "semantic_type": "", "description": "", "column": "e_y", "properties": {"dtype": "number", "std": 0.00022491044878129838, "min": -4.903011973709029e-06, "max": 0.000854469161381062, "num_unique_values": 3877, "samples": [0.00030226799618730295, 0.0008332061321181593, 0.0007319471781191474], "semantic_type": "", "description": "", "column": "Txy", "properties": {"dtype": "number", "std": 21678013.147140246, "min": -61295392.92830637, "max": 63149398.08348885, "num_unique_values": 3877, "samples": [9785230.53803435, -1267611.4796510637, 11866915.39012152], "semantic_type": "", "description": ""}}], "type": "dataframe", "variable_name": "data"}

```

```

plt.figure(figsize=(10, 6))
contour = plt.tricontourf(data['x'], data['y'], data['Txy'],

```

```
levels=20, cmap="jet")

# Step 2: Add a color bar for reference
cbar = plt.colorbar(contour)
cbar.set_label("Stress")

# Optional: Add labels, title, etc.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.title("Txy")

plt.show()
```

