In [8]:
```python
import sys
print(sys.executable)
print(sys.version)
```

```
c:\Users\rahul\OneDrive\Desktop\Notes\WS_2526\CS\Project_2_Computer_Experi
ment\Code\.venv\Scripts\python.exe
3.12.10 (tags/v3.12.10:0cc8128, Apr  8 2025, 12:21:36) [MSC v.1943 64 bit
(AMD64)]
```

In [21]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import joblib

from sklearn.model_selection import train_test_split, GridSearchCV, cross
from sklearn.metrics import r2_score, mean_squared_error, make_scorer

from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.pipeline import make_pipeline

from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, Matern, WhiteKernel, Co
```

In [ ]:
```python
# Load dataset
df = pd.read_csv("design_out.csv")

# Remove sigma_mem_y as it represents the membrane stress and is used to
# Not a model parametere which influences the model but helps to calculat
df = df[['f_mem','sigma_mem','E_mem','nu_mem','sigma_edg','sigma_sup','si
df.head() # 200 x 7
```

Out[ ]:

	f_mem	sigma_mem	E_mem	nu_mem	sigma_edg	sigma_s
0	0.268767	3220.880252	620830.965150	0.399472	325851.235578	322000.7623
1	0.511186	4603.969420	559505.773241	0.398297	266690.745363	366814.4072
2	0.316207	4020.464157	664063.602748	0.393333	350103.330214	308367.2050
3	0.457326	4282.802400	622714.404090	0.411230	313489.897816	327438.7028
4	0.261861	4097.033516	623600.545225	0.382459	334400.590712	474471.1278

In [10]:
```python
# Split into features (X) and target (y)
X = df.iloc[:, :-1]  # all columns except last
y = df.iloc[:, -1]   # last column = response

print("Features shape:", X.shape)
print("Target shape:", y.shape)

# print the features and response names
print("Response name:", y.name)
print("Feature names:", X.columns.tolist())
```

```
Features shape: (200, 6)
Target shape: (200,)
Response name: sigma_mem_max
Feature names: ['f_mem', 'sigma_mem', 'E_mem', 'nu_mem', 'sigma_edg', 'sig
ma_sup']
```

In [11]:
```
# Split the data into training and testing sets 80/20
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

1) SVR Model

In [ ]:
```
```

In [ ]:
```
# Build pipeline
# First standardize the data, then fit SVR model
pipeline = make_pipeline(StandardScaler(), SVR())

# Define search grid
param_grid = {
    "svr__C": [0.1, 1, 10, 100, 1000],
    "svr__epsilon": [0.001, 0.01, 0.1, 0.5, 1],
    "svr__gamma": ["scale", "auto", 0.001, 0.01, 0.1, 1]  # scale : (1 /
}

# Run grid search with 5-fold CV
grid = GridSearchCV(
    pipeline,
    param_grid,
    cv=KFold(n_splits=5, shuffle=True, random_state=42),
    scoring="r2",
    n_jobs=-1
)

grid.fit(X_train, y_train)

print("Best R2 Score:", grid.best_score_)
print("Best Parameters:", grid.best_params_)
```

```
Best R2 Score: 0.9917558451645704
Best Parameters: {'svr__C': 1000, 'svr__epsilon': 0.001, 'svr__gamma': 0.0
1}
```

In [ ]:
```
```

Out[ ]: 0.01

In [51]:
```
# Build final SVR model with optimal hyperparameters
svr = make_pipeline(
    StandardScaler(),
    SVR(kernel='rbf', C=grid.best_params_['svr__C'], epsilon=grid.best_pa
)

# Fit on the FULL dataset (not train/test split)
svr.fit(X_train, y_train)
```

```python
y_train_pred = svr.predict(X_train)
y_test_pred = svr.predict(X_test)
```

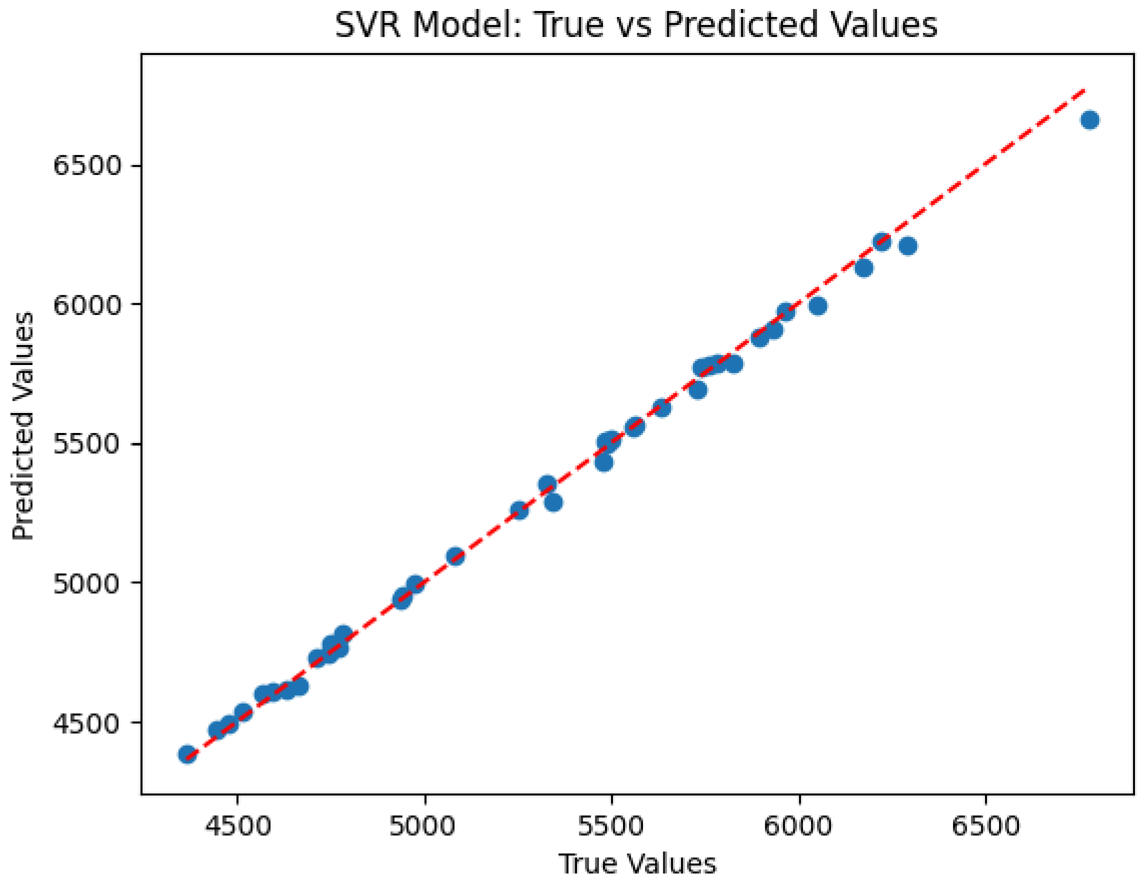
In [52]:
```python
# Training metrics
r2_train = r2_score(y_train, y_train_pred)
mse_train = mean_squared_error(y_train, y_train_pred)

# Test metrics
r2_test = r2_score(y_test, y_test_pred)
mse_test = mean_squared_error(y_test, y_test_pred)
```

In [54]:
```python
# Create datafram for errors storage with row names as model names and co
errors_df = pd.DataFrame(
    data={
        "R2_Train": [r2_train],
        "R2_Test": [r2_test],
        "MSE_Train": [mse_train],
        "MSE_Test": [mse_test]
    },
    index=["SVR"]
)

# Save the trained model
joblib.dump(svr, "svr.pkl")

errors_df
```

Out[54]:

	R2_Train	R2_Test	MSE_Train	MSE_Test
SVR	0.996683	0.997066	693.721558	1080.812744

In [56]:
```python
# Scatter plot of true vs predicted values
plt.scatter(y_test, y_test_pred)
plt.plot([y_test.min(), y_test.max()],
         [y_test.min(), y_test.max()], 'r--')
plt.xlabel("True Values")
plt.ylabel("Predicted Values")
plt.title("SVR Model: True vs Predicted Values")
plt.show()
```



2) Polynomial Regression

In [58]:
```
# List of degrees to try
degrees = [1, 2, 3, 4, 5, 6]

best_degree = None
best_mse = float('inf')
mse_scores_per_degree = []

for degree in degrees:
    # Create pipeline: polynomial features + scaling + linear regression
    model = make_pipeline(
        PolynomialFeatures(degree=degree),
        StandardScaler(),
        LinearRegression()
    )

    # 5-fold cross-validation for MSE (negative because cross_val_score m
    neg_mse_scores = cross_val_score(model, X_train, y_train, cv=KFold(n_
                                     scoring=make_scorer(mean_squared_err
    mse_scores = neg_mse_scores.mean()  # Average MSE across folds
    mse_scores_per_degree.append(mse_scores)

    print(f"Degree {degree}: Mean 5-CV MSE = {mse_scores:.4f}")

    # Keep track of the best degree
    if mse_scores < best_mse:
        best_mse = mse_scores
        best_degree = degree

print(f"\nBest degree based on 5-CV MSE: {best_degree}")
```

```python
# Fit final model with best degree on full training data
poly = make_pipeline(
    PolynomialFeatures(degree=best_degree),
    StandardScaler(),
    LinearRegression()
)

poly.fit(X_train, y_train)

# Predictions
y_train_pred = poly.predict(X_train)
y_test_pred = poly.predict(X_test)

# Training and test metrics

r2_train = r2_score(y_train, y_train_pred)
mse_train = mean_squared_error(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)
mse_test = mean_squared_error(y_test, y_test_pred)

print(f"\nTraining R²: {r2_train:.4f}, MSE: {mse_train:.4f}")
print(f"Test R²: {r2_test:.4f}, MSE: {mse_test:.4f}")
```

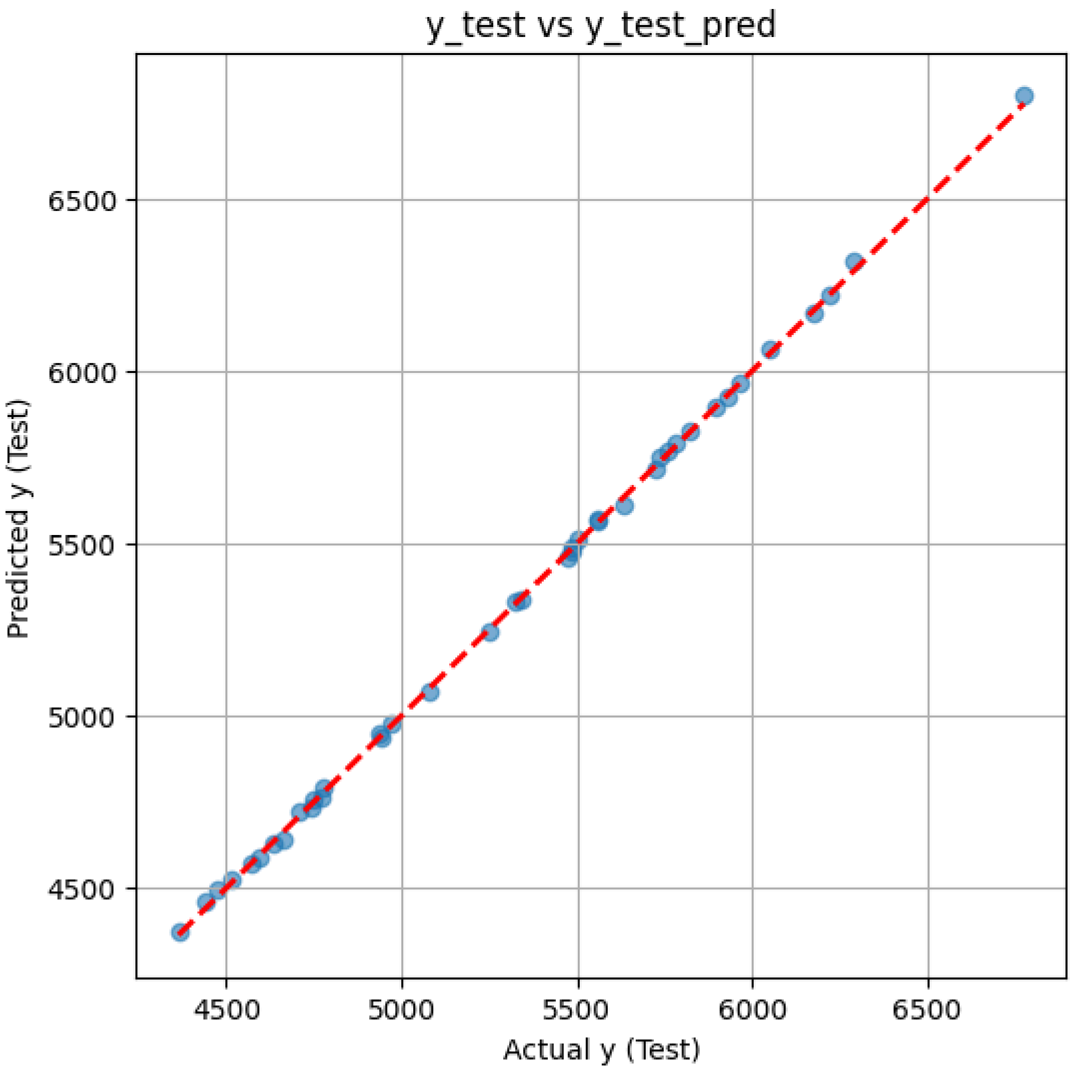
```
Degree 1: Mean 5-CV MSE = 3776.0046
Degree 2: Mean 5-CV MSE = 217.9354
Degree 3: Mean 5-CV MSE = 224.4187
Degree 4: Mean 5-CV MSE = 552.8711
Degree 5: Mean 5-CV MSE = 580.8720
Degree 6: Mean 5-CV MSE = 642.9481

Best degree based on 5-CV MSE: 2

Training R²: 0.9995, MSE: 110.2521
Test R²: 0.9996, MSE: 137.5833
```

In [59]:
```python
joblib.dump(poly, "poly.pkl")

# Errors from Polynomial Regression
errors_df.loc["PLY"] = [r2_train, r2_test, mse_train, mse_test]

# Display updated DataFrame
errors_df
```

Out[59]:

	R2_Train	R2_Test	MSE_Train	MSE_Test
SVR	0.996683	0.997066	693.721558	1080.812744
PLY	0.999473	0.999627	110.252095	137.583329

In [60]:
```python
plt.figure(figsize=(6,6))
plt.scatter(y_test, y_test_pred, alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--
plt.xlabel("Actual y (Test)")
plt.ylabel("Predicted y (Test)")
plt.title("y_test vs y_test_pred")
plt.grid(True)
plt.show()
```



3) GPR

In [86]:
```
# Scale features
scaler = StandardScaler()

# Define kernel: Constant * RBF
# kernel = C(1.0, (1e-3, 1e6)) * RBF(length_scale=1.0, length_scale_bound
kernel = 1 * RBF(length_scale=1.0, length_scale_bounds=(1e-3, 1e3)) + Whi
# kernel = 1 * RBF(length_scale=1.0, length_scale_bounds=(1e-3, 1e3))
# kernel = C(1.0, (1e-3, 1e3)) * Matern(length_scale=1.0, nu=1.5)




# Initialize Gaussian Process Regressor
gpr = make_pipeline(
    StandardScaler(),
    GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=10, norm
)

# Fit model
gpr.fit(X_train, y_train)

# print(gpr.kernel_)
# Predictions
```

```python
y_train_pred, y_train_std = gpr.predict(X_train, return_std=True)
y_test_pred, y_test_std = gpr.predict(X_test, return_std=True)

# Metrics
r2_train = r2_score(y_train, y_train_pred)
mse_train = mean_squared_error(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)
mse_test = mean_squared_error(y_test, y_test_pred)

print(f"Gaussian Process Regression -> Training R²: {r2_train:.4f}, MSE:
print(f"Gaussian Process Regression -> Test R²: {r2_test:.4f}, MSE: {mse_
print(gpr.named_steps['gaussianprocessregressor'].kernel_)
```

```
Gaussian Process Regression -> Training R²: 0.9997, MSE: 67.6423
Gaussian Process Regression -> Test R²: 0.9998, MSE: 80.2774
5.44**2 * RBF(length_scale=13.6) + WhiteKernel(noise_level=0.001)
```

```
c:\Users\rahul\OneDrive\Desktop\Notes\WS_2526\CS\Project_2_Computer_Experi
ment\Code\.venv\Lib\site-packages\sklearn\gaussian_process\kernels.py:440:
ConvergenceWarning: The optimal value found for dimension 0 of parameter k
2__noise_level is close to the specified lower bound 0.001. Decreasing the
bound and calling fit again may find a better value.
  warnings.warn(
```

In [87]:

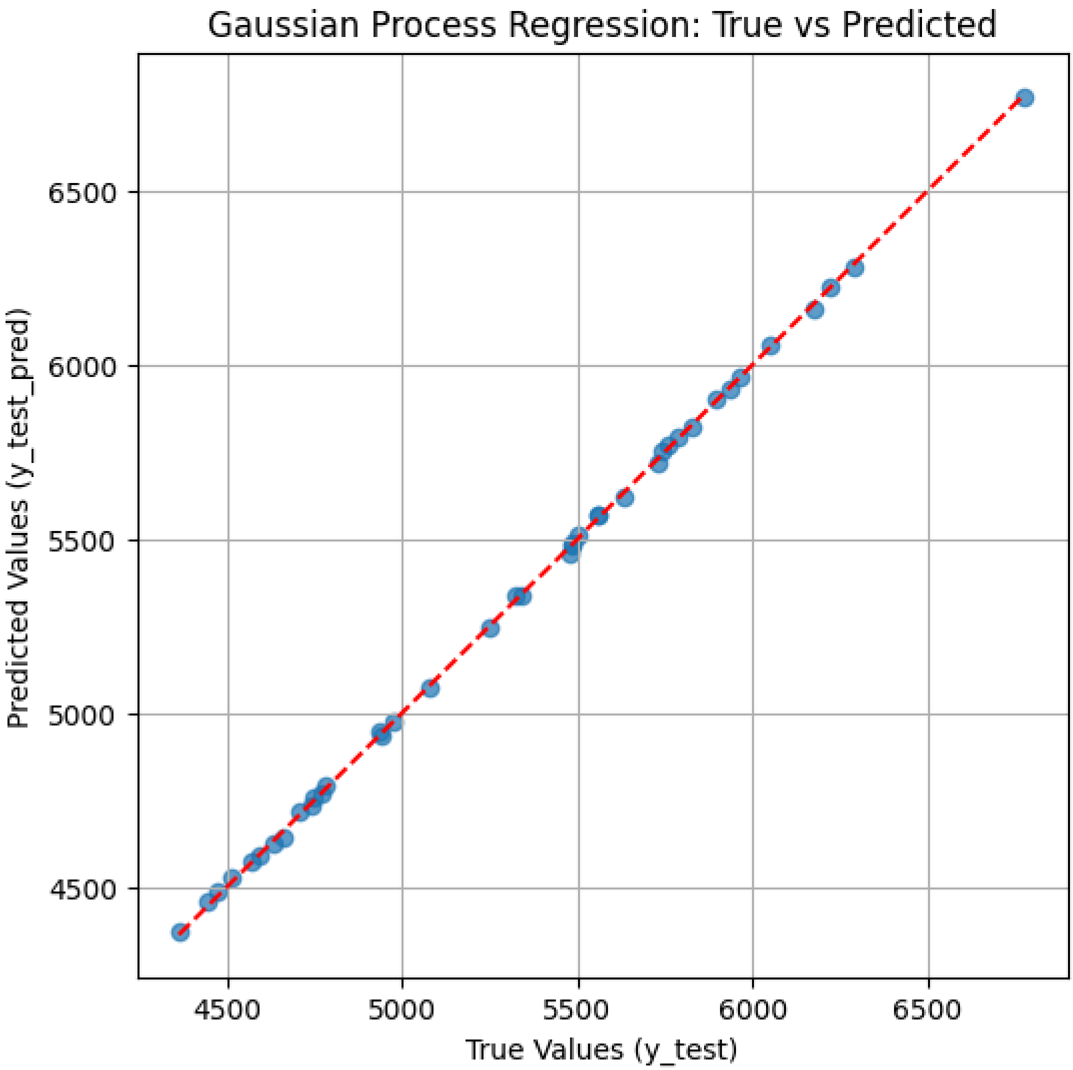
```python
joblib.dump(gpr, "gpr.pkl")

# Errors from Gaussian Regression
errors_df.loc["GPR"] = [r2_train, r2_test, mse_train, mse_test]

# Display updated DataFrame
errors_df
```

Out[87]:

	R2_Train	R2_Test	MSE_Train	MSE_Test
SVR	0.996683	0.997066	693.721558	1080.812744
PLY	0.999473	0.999627	110.252095	137.583329
GPR	0.999677	0.999782	67.642341	80.277439

In [88]:

```python
plt.figure(figsize=(6, 6))
plt.scatter(y_test, y_test_pred, alpha=0.7)
plt.plot([y_test.min(), y_test.max()],
         [y_test.min(), y_test.max()],
         linestyle='--', color='r')
plt.xlabel("True Values (y_test)")
plt.ylabel("Predicted Values (y_test_pred)")
plt.title("Gaussian Process Regression: True vs Predicted")
plt.grid(True)
plt.show()
```



4) Uncertainty Propagation

In [89]:
```
import numpy as np
import pandas as pd

# Constants
N = 10000
gamma = 0.5772156649  # Euler-Mascheroni constant

def get_lognormal_params(mean, sd):
    var = sd**2
    sigma = np.sqrt(np.log(1 + var / mean**2))
    mu = np.log(mean) - 0.5 * sigma**2
    return mu, sigma

def get_gumbel_params(mean, sd):
    scale = sd * np.sqrt(6) / np.pi
    loc = mean - scale * gamma
    return loc, scale

def get_uniform_params(mean, sd):
    # SD = (high - low) / sqrt(12)
    # Mean = (high + low) / 2
    delta = sd * np.sqrt(3)
    low = mean - delta
```

```python
    high = mean + delta
    return low, high

# Generate data
data = {}

# f_mem: Gumbel
loc, scale = get_gumbel_params(0.4, 0.12)
data['f_mem'] = np.random.gumbel(loc, scale, N)

# Lognormal variables
lognorm_vars = {
    'sigma_mem_y': (11000, 1650),
    'sigma_mem': (4000, 800),
    'E_mem': (600000, 90000),
    'sigma_edg': (353677.6513, 70735.53026),
    'sigma_sup': (400834.6715, 80166.9343)
}

for name, (m, s) in lognorm_vars.items():
    mu, sigma = get_lognormal_params(m, s)
    data[name] = np.random.lognormal(mu, sigma, N)

# nu_mem: Uniform
low, high = get_uniform_params(0.4, 0.01154700538)
data['nu_mem'] = np.random.uniform(low, high, N)

try:
    samples_df = pd.read_csv('sample_points.csv')
except FileNotFoundError:
    # Create DataFrame
    samples_df = pd.DataFrame(data)
    # Save to CSV
    samples_df.to_csv('sample_points.csv', index=False)

# Summary statistics for verification
summary = samples_df.agg(['mean', 'std']).T
summary['variance'] = summary['std']**2
print(summary)
```

```
                      mean           std      variance
f_mem             0.400017      0.122363  1.497269e-02
sigma_mem_y   11006.915552   1647.463875  2.714137e+06
sigma_mem      4004.092529    800.288791  6.404621e+05
E_mem        599744.298013  91200.170008  8.317471e+09
sigma_edg    354031.078148  70842.419793  5.018648e+09
sigma_sup    400039.964252  81926.101818  6.711886e+09
nu_mem            0.400013      0.011426  1.305524e-04
```

In [90]:
```python
samples_df = samples_df[['f_mem','sigma_mem','E_mem','nu_mem','sigma_edg'
samples_df.head()
samples = samples_df.values
```

In [91]:
```python
poly.predict(samples[0:10])
```

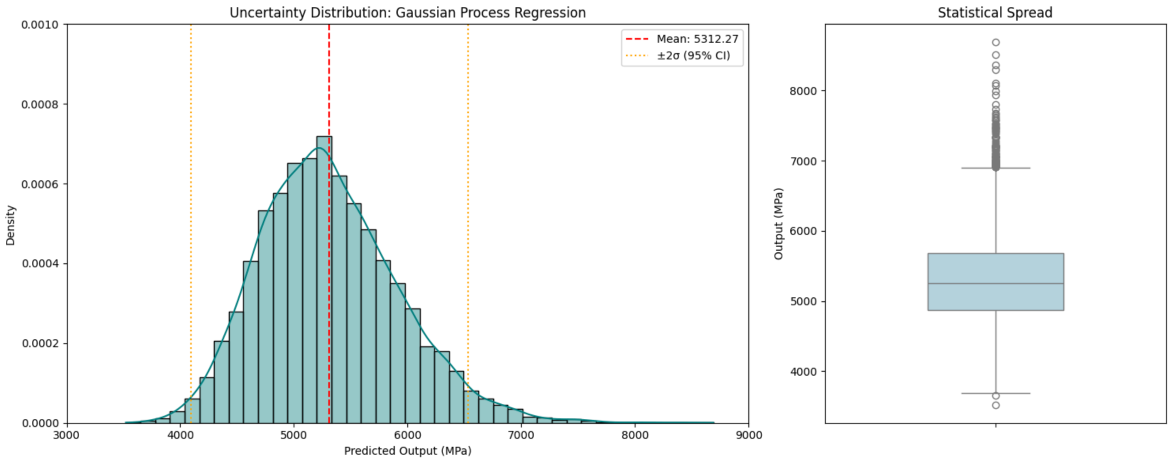
```
c:\Users\rahul\OneDrive\Desktop\Notes\WS_2526\CS\Project_2_Computer_Experi
ment\Code\.venv\Lib\site-packages\sklearn\utils\validation.py:2691: UserWa
rning: X does not have valid feature names, but PolynomialFeatures was fit
ted with feature names
  warnings.warn(
```

```
Out[91]: array([5490.28901174, 5601.69053691, 4330.12129635, 6036.37014966,
                6903.1560084 , 4497.79035799, 5344.15778607, 4686.5053377 ,
                5965.08856618, 5568.03075001])
```

```
In [93]: import numpy as np
         import matplotlib.pyplot as plt
         import seaborn as sns

         def uncertainty(model, samples, model_name="Surrogate Model", unit="Units
             """
             Performs uncertainty quantification on a model using provided samples

             Parameters:
             model: Trained sklearn-compatible model (SVR, GP, etc.)
             samples: (N, features) array of Monte Carlo samples
             model_name: String for plot titles
             unit: String for axis labels (e.g., 'MPa' or 'kN')
             """
             # 1. Generate Predictions
             predictions = model.predict(samples).flatten()

             # 2. Calculate Statistics
             mu = np.mean(predictions)
             sigma = np.std(predictions)
             cv = (sigma / mu) * 100 if mu != 0 else 0  # Coefficient of Variation

             # 3. Create Visualization
             fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6), gridspec_kw={'w

             # --- Plot 1: Histogram & KDE ---
             sns.histplot(predictions, kde=True, ax=ax1, color='teal', bins=40, st
             ax1.axvline(mu, color='red', linestyle='--', label=f'Mean: {mu:.2f}')
             ax1.axvline(mu - 2*sigma, color='orange', linestyle=':', label=f'±2σ
             ax1.axvline(mu + 2*sigma, color='orange', linestyle=':')

             ax1.set_title(f"Uncertainty Distribution: {model_name}")
             ax1.set_xlabel(f"Predicted Output ({unit})")
             ax1.set_xlim(3000, 9000)
             ax1.set_ylim(0, 0.001)
             ax1.legend()

             # --- Plot 2: Box & Whisker (Outlier Analysis) ---
             sns.boxplot(y=predictions, ax=ax2, color='lightblue', width=0.4)
             ax2.set_title("Statistical Spread")
             ax2.set_ylabel(f"Output ({unit})")

             plt.tight_layout()
             plt.show()

             # 4. Return Summary Report
             return {
                 "Mean": mu,
                 "Std_Dev": sigma,
                 "COV_Percent": cv,
                 "Min": np.min(predictions),
                 "Max": np.max(predictions)
             }
```

```
In [95]: uncertainty(gpr, samples, model_name="Gaussian Process Regression", unit=
```

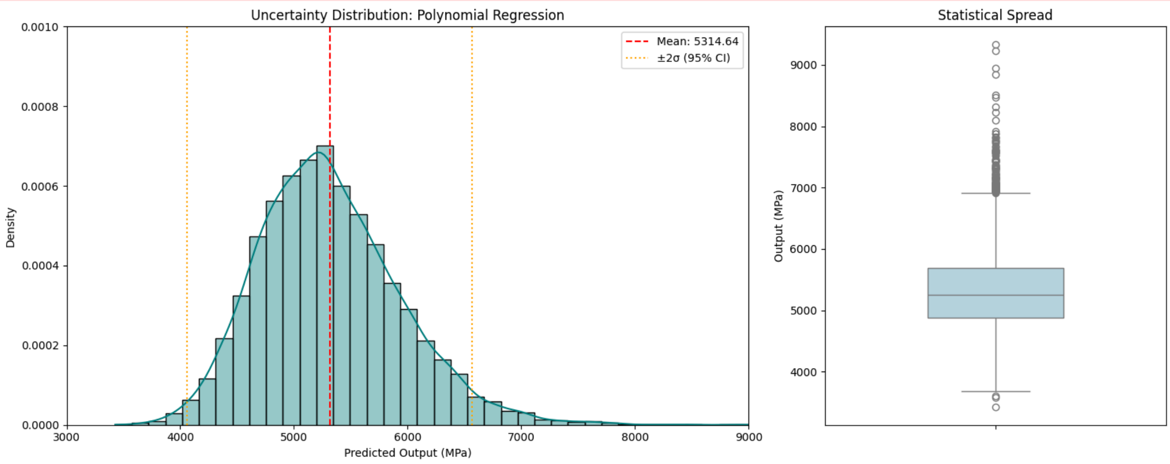
```
c:\Users\rahul\OneDrive\Desktop\Notes\WS_2526\CS\Project_2_Computer_Experiment\Code\.venv\Lib\site-packages\sklearn\utils\validation.py:2691: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```



Out[95]:
```
{'Mean': np.float64(5312.2654550072075),
 'Std_Dev': np.float64(609.9888347606692),
 'COV_Percent': np.float64(11.482649726882702),
 'Min': np.float64(3520.9951222202635),
 'Max': np.float64(8689.606110066255)}
```

In [94]:
```
uncertainty(poly, samples, model_name="Polynomial Regression", unit="MPa"
```

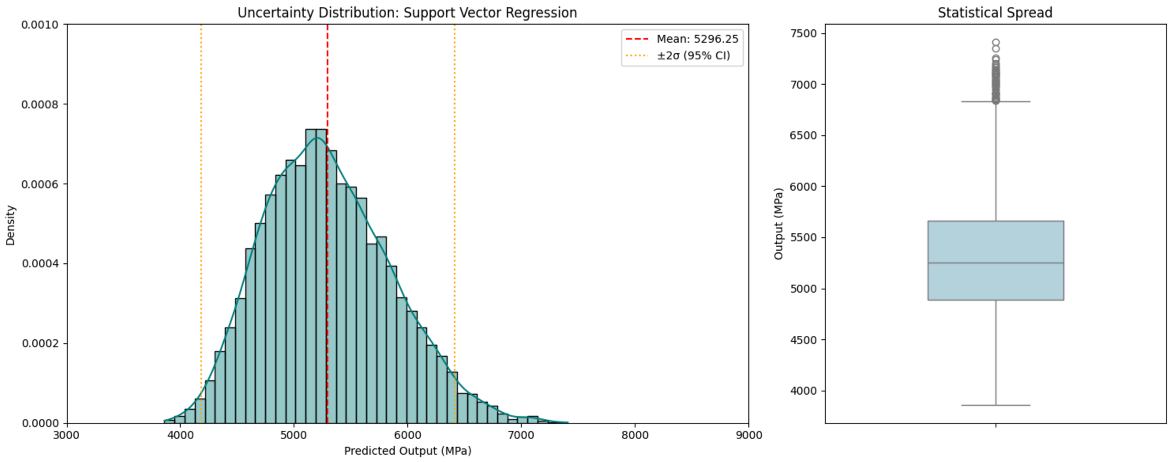
```
c:\Users\rahul\OneDrive\Desktop\Notes\WS_2526\CS\Project_2_Computer_Experiment\Code\.venv\Lib\site-packages\sklearn\utils\validation.py:2691: UserWarning: X does not have valid feature names, but PolynomialFeatures was fitted with feature names
  warnings.warn(
```



Out[94]:
```
{'Mean': np.float64(5314.635055261468),
 'Std_Dev': np.float64(626.7527882294062),
 'COV_Percent': np.float64(11.792960037941333),
 'Min': np.float64(3427.106456259691),
 'Max': np.float64(9329.472370064675)}
```

In [ ]:
```
uncertainty(svr, samples, model_name="Support Vector Regression", unit="M
```

```
/Users/anirudhparameswaran/Desktop/Case Studies - Computer Experiments/SVM/csenv/lib/python3.14/site-packages/sklearn/utils/validation.py:2749: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```




```
Out[ ]: {'Mean': np.float64(5296.252812601258),
 'Std_Dev': np.float64(557.4042996804361),
 'COV_Percent': np.float64(10.524503255475576),
 'Min': np.float64(3860.85860081164),
 'Max': np.float64(7409.324511311017)}
```

```
In [ ]:
```