Lab 8

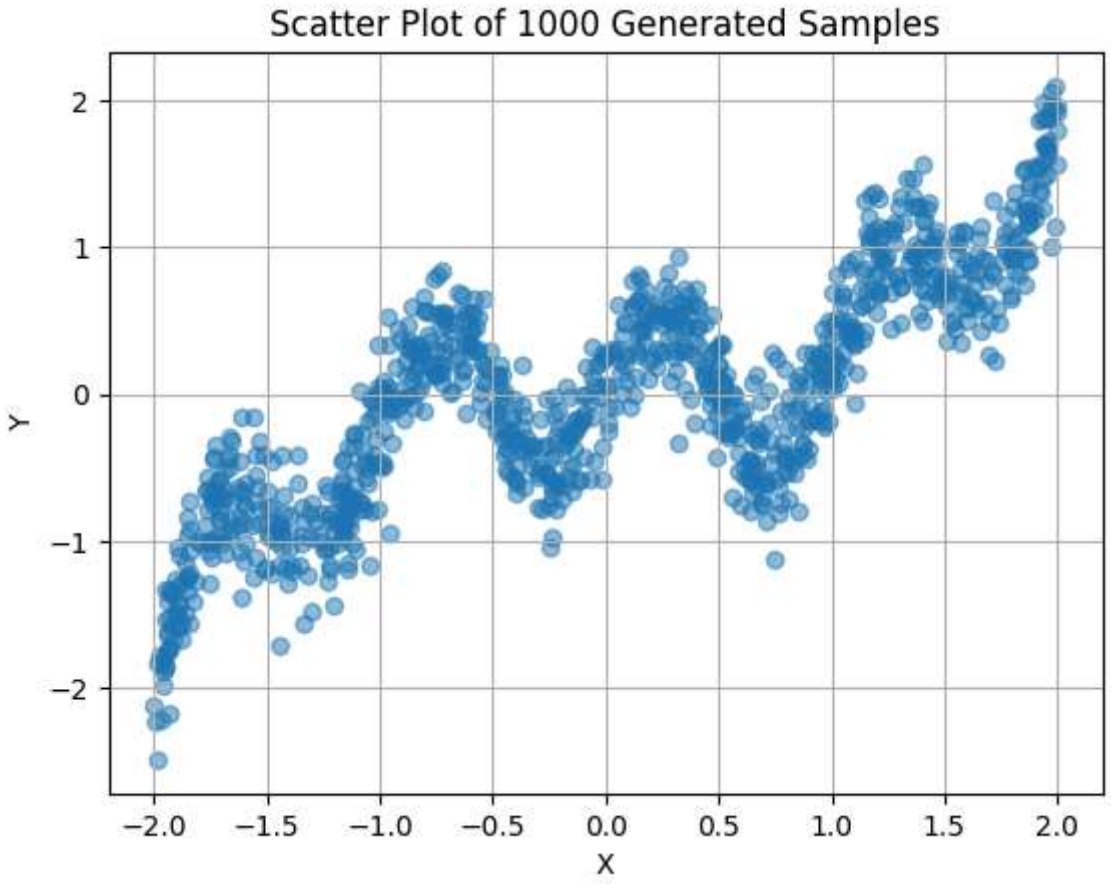
In [ ]:
```
ID = 1631938
Name = 'MIN SOE HTUT'
```

2 Define the data generator

In [ ]:
```
import numpy as np

def ground_truth_concept(x):
    y=0.25*x**3+0.5*np.sin(np.pi*2*x)
    return y

def data_generator(n_examples,seed=314159265):
  np.random.seed(seed)
  x=(np.random.rand(n_examples)-0.5)*4
  np.random.seed(seed+1)
  y=ground_truth_concept(x)+np.random.randn(n_examples)*0.25
  x=x.reshape(n_examples,1)
  return x,y
```

3 Generate 1000 samples using data_generator and visualize it using a scatter plot

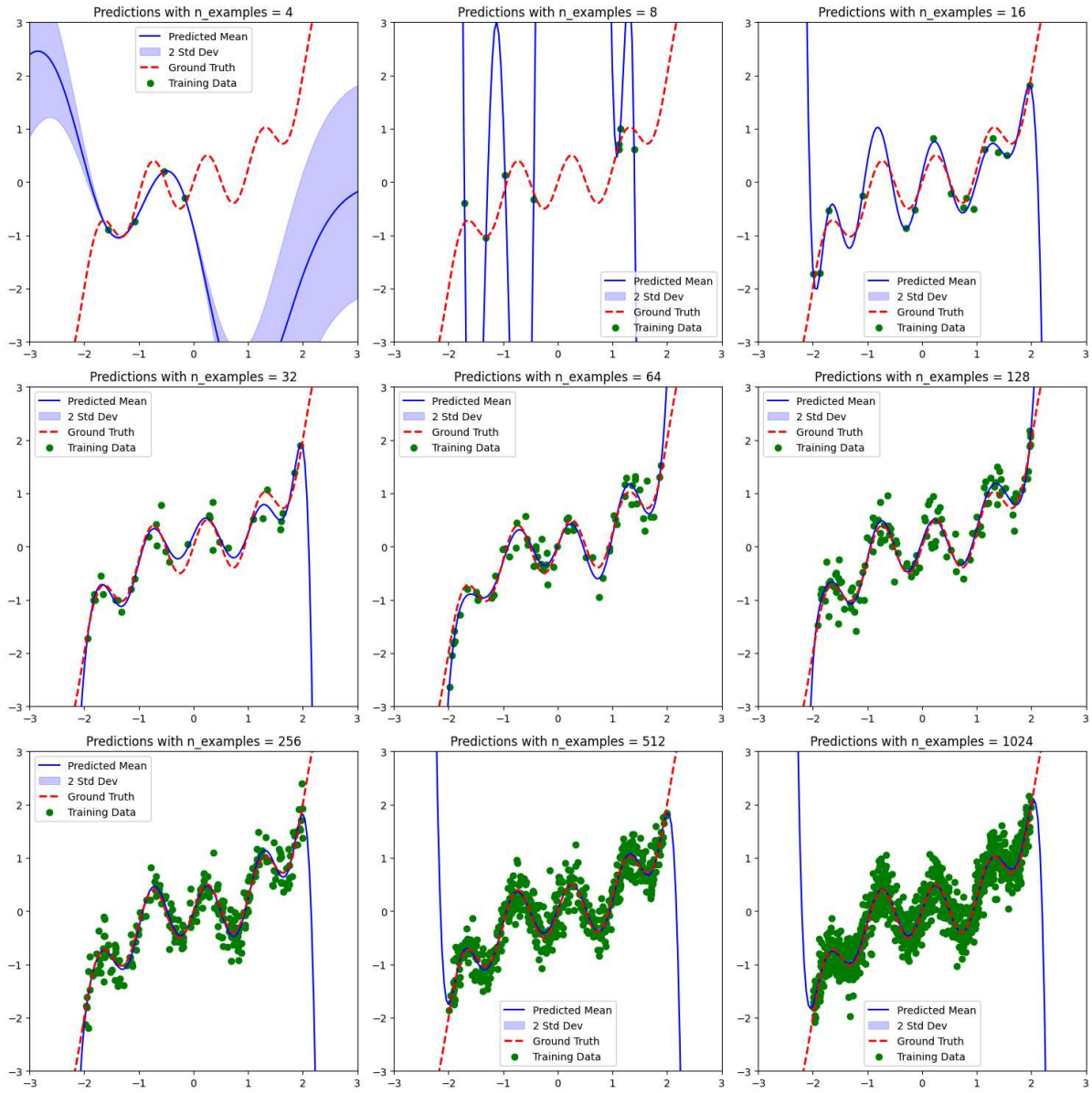
In [ ]:
```
import matplotlib.pyplot as plt

# Generate 1000 samples using data_generator
np.random.seed(ID)
x_samples, y_samples = data_generator(1000, ID)

# Plotting the data using a scatter plot
plt.scatter(x_samples, y_samples, alpha=0.5)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Scatter Plot of 1000 Generated Samples')
plt.grid(True)
plt.show()
```



4 Do a 3x3 subplot, in each subplot, you will plot the graphs for one of each results of for n_examples=4,8,16,32,64,128,256,512,1024

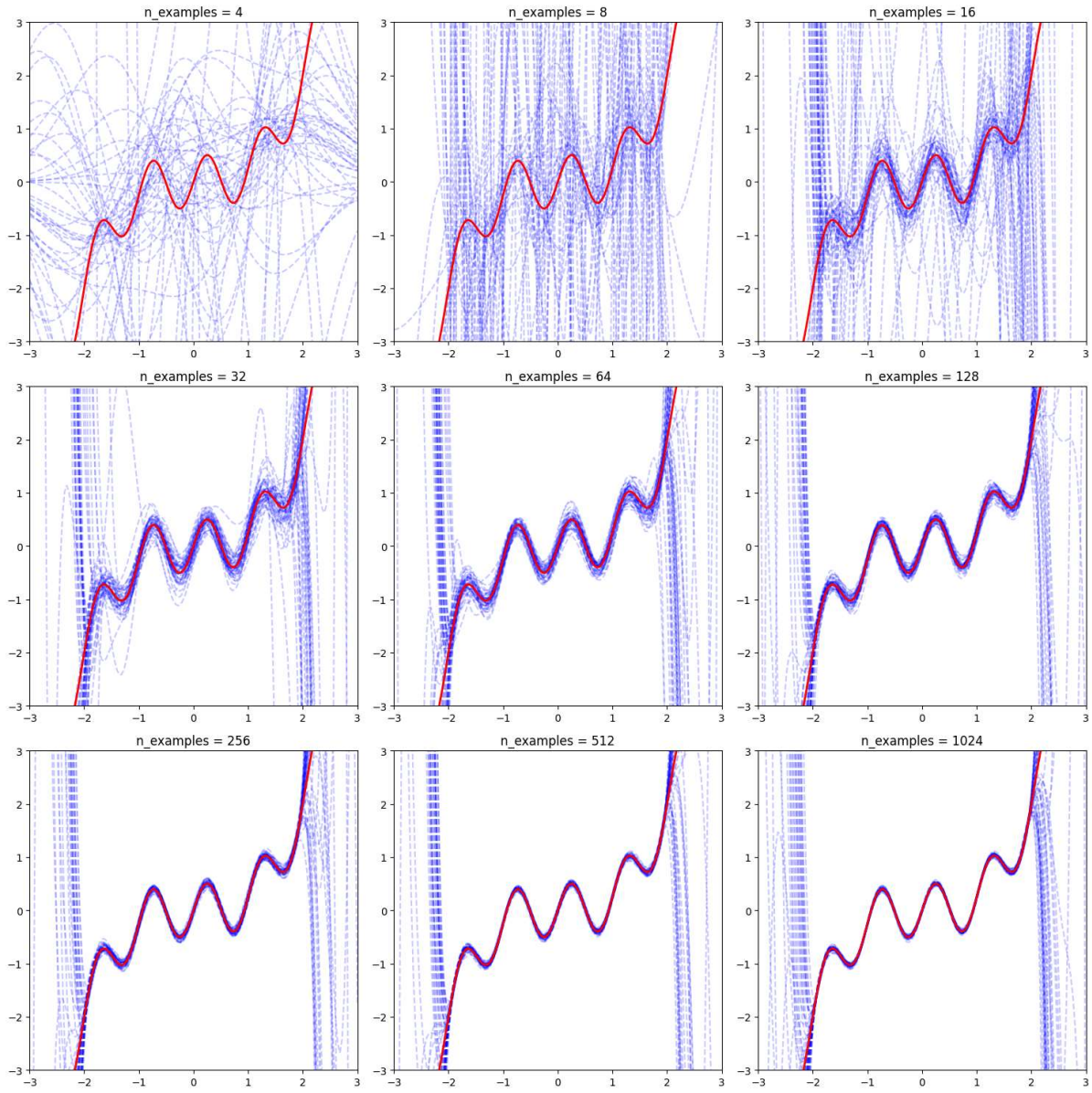
In [ ]:
```
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF
import numpy as np
import matplotlib.pyplot as plt

# Define n_examples values as specified
n_examples_values = [4, 8, 16, 32, 64, 128, 256, 512, 1024]

# Set up a 3x3 subplot layout
fig, axes = plt.subplots(3, 3, figsize=(15, 15))

# Define X_test to evaluate the predictions on a fine grid
X_test = np.linspace(-3, 3, 151).reshape(-1, 1)
y_test = ground_truth_concept(X_test)

# Loop through each n_examples value and generate subplots
for i, n_examples in enumerate(n_examples_values):
    # Get the current subplot
    ax = axes[i // 3, i % 3]

    # Generate the data
    x_data, y_data = data_generator(n_examples, ID + i)

    # Fit the GaussianProcessRegressor (with the default RBF kernel)
    gp = GaussianProcessRegressor()
    gp.fit(x_data, y_data)

    # Make predictions on X_test and get the standard deviations
    y_pred, sigma = gp.predict(X_test, return_std=True)

    # Plot the predicted mean
    ax.plot(X_test, y_pred, 'b-', label='Predicted Mean')

    # Plot the 2 standard deviation intervals
    ax.fill_between(X_test[:, 0], y_pred - 2 * sigma, y_pred + 2 * sigma, colo
r='blue', alpha=0.2, label='2 Std Dev')

    # Overlay the ground truth concept
    ax.plot(X_test, y_test, 'r--', label='Ground Truth', linewidth=2)

    # Scatter plot of the training data
    ax.scatter(x_data, y_data, color='green', marker='o', label='Training Dat
a')

    # Set the plot limits for x and y
    ax.set_xlim(-3, 3)
    ax.set_ylim(-3, 3)

    # Set the title for the current subplot
    ax.set_title(f'Predictions with n_examples = {n_examples}')

    # Add the legend for all subplots
    ax.legend()

# Adjust layout to prevent overlapping subplots
plt.tight_layout()
```

```
# Show the 3x3 subplots
plt.show()
```



5. Do a 3x3 subplot, in each subplot, you will plot the graphs for one of each results for n_examples=4,8,16,32,64,128,256,512,1024

In [ ]:
```
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF
import numpy as np
import matplotlib.pyplot as plt

# Define n_examples values
n_examples_values = [4, 8, 16, 32, 64, 128, 256, 512, 1024]

# Set up a 3x3 subplot layout
fig, axes = plt.subplots(3, 3, figsize=(15, 15))

# Define X_test for prediction and the ground truth concept
X_test = np.linspace(-3, 3, 151).reshape(-1, 1)
y_test = ground_truth_concept(X_test)

# Loop through each n_examples value and generate subplots
for i, n_examples in enumerate(n_examples_values):
    # Get the current subplot
    ax = axes[i // 3, i % 3]

    # Repeat the data generation and GPR fitting 50 times
    for j in range(50):
        # Generate data for each iteration (vary the seed for randomness)
        x_data, y_data = data_generator(n_examples, ID + i * 50 + j)

        # Fit the GaussianProcessRegressor (with the default RBF kernel)
        gp = GaussianProcessRegressor()
        gp.fit(x_data, y_data)

        # Make predictions on X_test
        y_pred = gp.predict(X_test)

        # Plot each mean prediction as a dashed line (set alpha to make them s
emi-transparent)
        ax.plot(X_test, y_pred, 'b--', alpha=0.2)

    # Overlay the ground truth concept in red
    ax.plot(X_test, y_test, 'r-', linewidth=2, label='Ground Truth')

    # Set the plot limits for x and y
    ax.set_xlim(-3, 3)
    ax.set_ylim(-3, 3)

    # Set the title for the current subplot
    ax.set_title(f'n_examples = {n_examples}')

# Adjust layout to prevent overlapping subplots
plt.tight_layout()

# Show the 3x3 subplots
plt.show()
```



Part B

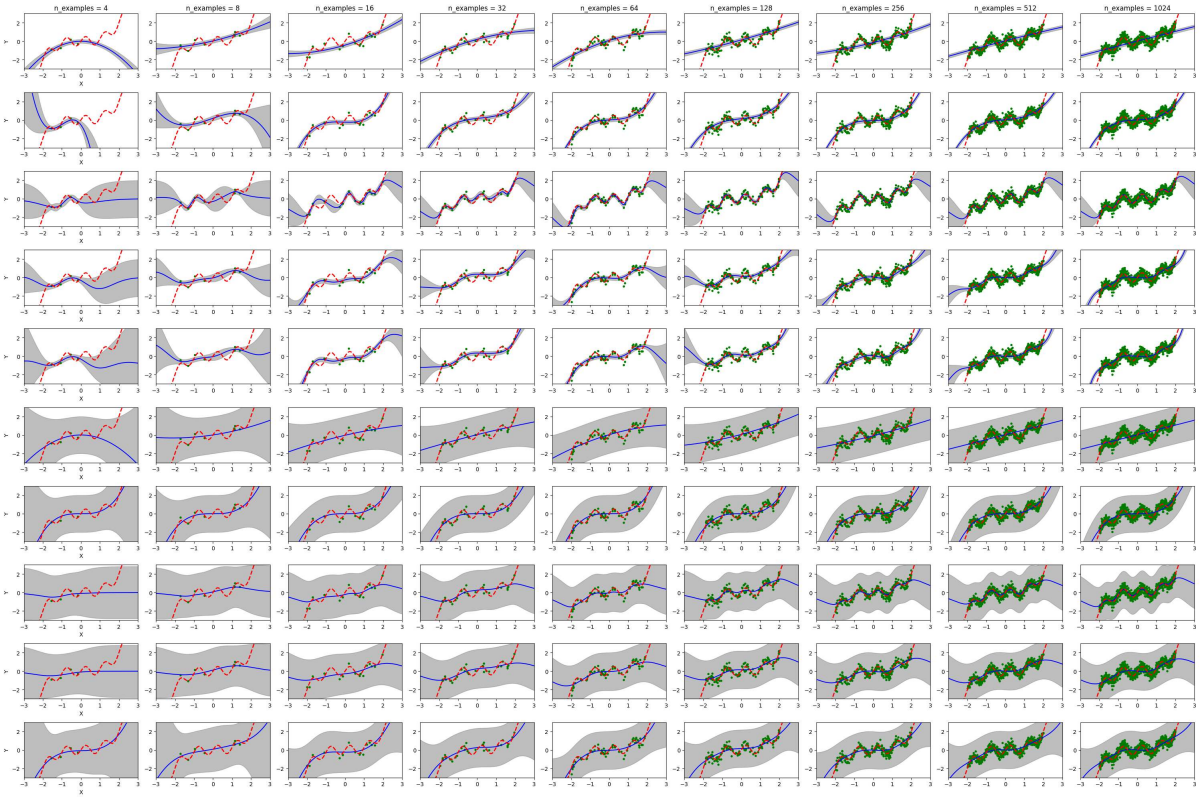
1) Define the Kernels

In [26]:
```
from sklearn.gaussian_process.kernels import DotProduct, WhiteKernel, Matern,
RBF, Exponentiation

# Define the kernels
kernels = [
    Exponentiation(DotProduct(), 2) + WhiteKernel(noise_level=0.01, noise_leve
l_bounds='fixed'),  # Polynomial degree 2 + fixed White noise 0.01
    Exponentiation(DotProduct(), 3) + WhiteKernel(noise_level=0.01, noise_leve
l_bounds='fixed'),  # Polynomial degree 3 + fixed White noise 0.01
    Matern(length_scale=1.0, length_scale_bounds='fixed') + WhiteKernel(noise_
level=0.01, noise_level_bounds='fixed'),  # Matern with fixed length_scale
    RBF(length_scale=1.0, length_scale_bounds='fixed') + WhiteKernel(noise_lev
el=0.01, noise_level_bounds='fixed'),  # RBF with fixed length_scale
    Exponentiation(DotProduct(), 2) + RBF(length_scale=1.0, length_scale_bound
s='fixed') + WhiteKernel(noise_level=0.01, noise_level_bounds='fixed'),  # Pol
ynomial degree 2 + RBF + fixed length_scale
    Exponentiation(DotProduct(), 2) + WhiteKernel(noise_level=1.0, noise_level
_bounds='fixed'),  # Polynomial degree 2 + fixed White noise 1.0
    Exponentiation(DotProduct(), 3) + WhiteKernel(noise_level=1.0, noise_level
_bounds='fixed'),  # Polynomial degree 3 + fixed White noise 1.0
    Matern(length_scale=1.0, length_scale_bounds='fixed') + WhiteKernel(noise_
level=1.0, noise_level_bounds='fixed'),  # Matern with fixed length_scale
    RBF(length_scale=1.0, length_scale_bounds='fixed') + WhiteKernel(noise_lev
el=1.0, noise_level_bounds='fixed'),  # RBF with fixed length_scale
    Exponentiation(DotProduct(), 3) + RBF(length_scale=1.0, length_scale_bound
s='fixed') + WhiteKernel(noise_level=1.0, noise_level_bounds='fixed')  # Polyn
omial degree 3 + RBF + fixed length_scale
]
```

2) For each of the kernels in part B.1)Doing a 3x3 subplot, in each subplot, you will plot the graphs for one of each results of for n_examples=4,8,16,32,64,128,256,512,1024

In [27]:
```
from sklearn.gaussian_process import GaussianProcessRegressor
import numpy as np
import matplotlib.pyplot as plt

# Define X_test for generating predictions and the ground truth
X_test = np.linspace(-3, 3, 151).reshape(-1, 1)
y_test = 0.25 * X_test**3 + 0.5 * np.sin(np.pi * 2 * X_test)

# Define n_examples values as specified
n_examples_values = [4, 8, 16, 32, 64, 128, 256, 512, 1024]

# Kernel names for easy identification in the plots
kernel_names = [
    "Poly (deg 2) + White noise 0.01",
    "Poly (deg 3) + White noise 0.01",
    "Matern + White noise 0.01",
    "RBF + White noise 0.01",
    "Poly (deg 2) + RBF + White noise 0.01",
    "Poly (deg 2) + White noise 1.0",
    "Poly (deg 3) + White noise 1.0",
    "Matern + White noise 1.0",
    "RBF + White noise 1.0",
    "Poly (deg 3) + RBF + White noise 1.0"
]

# Function to generate training data (based on a provided seed)
def data_generator(n_examples, seed):
    np.random.seed(seed)
    x = (np.random.rand(n_examples) - 0.5) * 4  # Random x-values in range (-
2, 2)
    np.random.seed(seed + 1)
    y = 0.25 * x**3 + 0.5 * np.sin(np.pi * 2 * x) + np.random.randn(n_example
s) * 0.25  # Noisy data
    x = x.reshape(n_examples, 1)
    return x, y

# Function to plot for each kernel
def plot_for_kernel(kernel, kernel_name, axes, row_idx):
    # Loop through each n_examples value
    for i, n_examples in enumerate(n_examples_values):
        ax = axes[row_idx, i]

        # Generate the data
        x_data, y_data = data_generator(n_examples, ID + i)

        # Fit the GaussianProcessRegressor with the current kernel
        gp = GaussianProcessRegressor(kernel=kernel)
        gp.fit(x_data, y_data)

        # Predict the mean and standard deviations
        y_pred, sigma = gp.predict(X_test, return_std=True)

        # Plot the mean predictions
        ax.plot(X_test, y_pred, 'b-', label='Predicted Mean')

        # Plot the 2 standard deviation intervals
```

```
        ax.fill_between(X_test.flatten(), y_pred - 2 * sigma, y_pred + 2 * sig
ma, color='gray', alpha=0.5, label='2 Std Dev')

        # Overlay the ground truth concept
        ax.plot(X_test, y_test, 'r--', label='Ground Truth', linewidth=2)

        # Scatter plot of the training data
        ax.scatter(x_data, y_data, color='green', marker='.', label='Training
Data')

        # Set plot limits
        ax.set_xlim(-3, 3)
        ax.set_ylim(-3, 3)

        # Set the title for the current subplot
        if row_idx == 0:
            ax.set_title(f'n_examples = {n_examples}')

    # Add labels and legend only to the first column and row
    if row_idx == len(kernels) - 1:
        for ax in axes[:, 0]:
            ax.set(xlabel='X', ylabel='Y')

# Create the large figure for all subplots
fig, axes = plt.subplots(len(kernels), len(n_examples_values), figsize=(30, 2
0))

# Loop through all kernels and generate plots
for idx, kernel in enumerate(kernels):
    plot_for_kernel(kernel, kernel_names[idx], axes, idx)

# Ensure the layout is tight and not overlapping
plt.tight_layout()
plt.show()
```



Part C

a) The behavoir of the regressors as the number of examples used to estimate the regressors increase

As seen in the plots, when the number of examples is small n_examples = 4 or n_examples = 8 the model exhibits large uncertainty shown by wider 2-standard-deviation intervals. The predictions fluctuate and deviate more significantly from the ground truth indicating that the model has insufficient data to make confident predictions. As the number of examples increases the predictions become more accurate and the uncertainty bands narrow. By n_examples = 1024, the regressors closely fit the ground truth, and the variance in predictions decreases dramatically. This demonstrates the model's improved confidence with more training data.

b) The extrapolation (the estimated curve outside the region of the data) behavior of the regressors

The extrapolation behavior is highly dependent on the kernel.We observe that the RBF and Matern kernels extrapolate more smoothly outside the range of the training data. These kernels revert to the mean and produce reasonable consistent predictions outside the data region. Polynomial kernels with higher degrees exhibit more erratic behavior when extrapolating showing large deviations from the ground truth in the extrapolated regions. This is particularly evident when the data is sparse where polynomial kernels tend to overfit within the data range and fail to generalize beyond it.

c) The ability for the regressor to "capture" the wriggly behaivoir in the middle of the data

The RBF and Matern kernels perform well in capturing the wriggly behavior in the middle of the data .These kernels are designed to handle smooth and continuous variations, which makes them well-suited to model the oscillations in the data.Polynomial kernels especially lower-degree ones struggle to capture these local variations. They impose global trends leading to a more linear or smooth fit that does not accurately reflect the wriggly nature of the data. Higher degree polynomials can capture some of the wiggles but are prone to overfitting particularly when the data is sparse.

d) The amount of estimated (white) noise epsilon

The white noise component significantly affects the model ability to handle deviations in the data.Models with higher noise levels show wider uncertainty intervals indicating that the model is accounting for more stochastic variation. This results in smoother predictions as the model does not attempt to fit every small fluctuation in the data. When the noise level is small or fixed the model fits the data more tightly reducing the apparent noise in the predictions. This is evident in the plots where smaller noise levels lead to tighter confidence intervals and more precise fits.