Lab 6: Gaussian Processes - Modeling and Predicting CO2 measurements

The goals of this lab are:

- Practice using Gaussian processes
- Compare the use of different kernels
- Model CO2 measurements in the atmosphere and make predictions on the CO2 levels in the future

In this lab, we will be using data from the the Mauna Loa Observatory, which has been making CO2 measurements of the atmopshere on top of Maunaloa since the late 1950's. It has one of the longest consistent set of measurements of CO2 measurements ever made. It is one of the sources of data used to model climate change. More information on the measurements can be found at: https://www.esrl.noaa.gov/gmd/ccgg/about/co2_measurements.html

Created by: Tuan Do

Last modified: Tuan Do

In [1]:
```
# imports here
import numpy as np
import pandas as pd
import pylab as plt

import matplotlib
font = {        'size'   : 20}
matplotlib.rc('font', **font)
```

Part 1 Big Picture

Using the CO2 measurements that have been made in the past, we want to predict CO2 measurements in the future as well as the uncertainty in the prediction. We also want to differentiate between seasonal variability from long term trends in the data.

We have two major questions to address in this lab:

1. **How far into the future can we reliably predict the CO2 level of the atmosphere using Gaussian Processes?**

2. **Did the CO2 level rise slow due to the global COVID19 pandemic?**

Part 2 Get the data

The Mauna Loa Observatory makes their data downloadable here: https://www.esrl.noaa.gov/gmd/ccgg/trends/data.html

We've already downloaded the monthly averaged data into your Lab 6 directory. Load it using the following code.

We will use two of the columns for our work: `decimal date` and `average`, which contains the decimal year for the time and the monthly averaged CO2 concentration in parts per million molecules.

In [2]:
```
# load the data
tab = pd.read_csv('co2_mm_mlo.csv',comment='#')
```

Part 3 Explore the data

Question 1

(3 pts)

Check the shape, and head, plot, visualize, and have a look at the data with whatever ways you think will help your task. What's the time span of the dataset? Etc. Comment throughout on observations.

In [3]:
```
# code here
tab.shape
```

Out[3]:
```
(791, 8)
```

In [4]:
```
tab.head()
```

Out[4]:

	year	month	decimal date	average	deseasonalized	ndays	sdev	unc
0	1958	3	1958.2027	315.70	314.43	-1	-9.99	-0.99
1	1958	4	1958.2877	317.45	315.16	-1	-9.99	-0.99
2	1958	5	1958.3699	317.51	314.71	-1	-9.99	-0.99
3	1958	6	1958.4548	317.24	315.14	-1	-9.99	-0.99
4	1958	7	1958.5370	315.86	315.18	-1	-9.99	-0.99

In [5]:

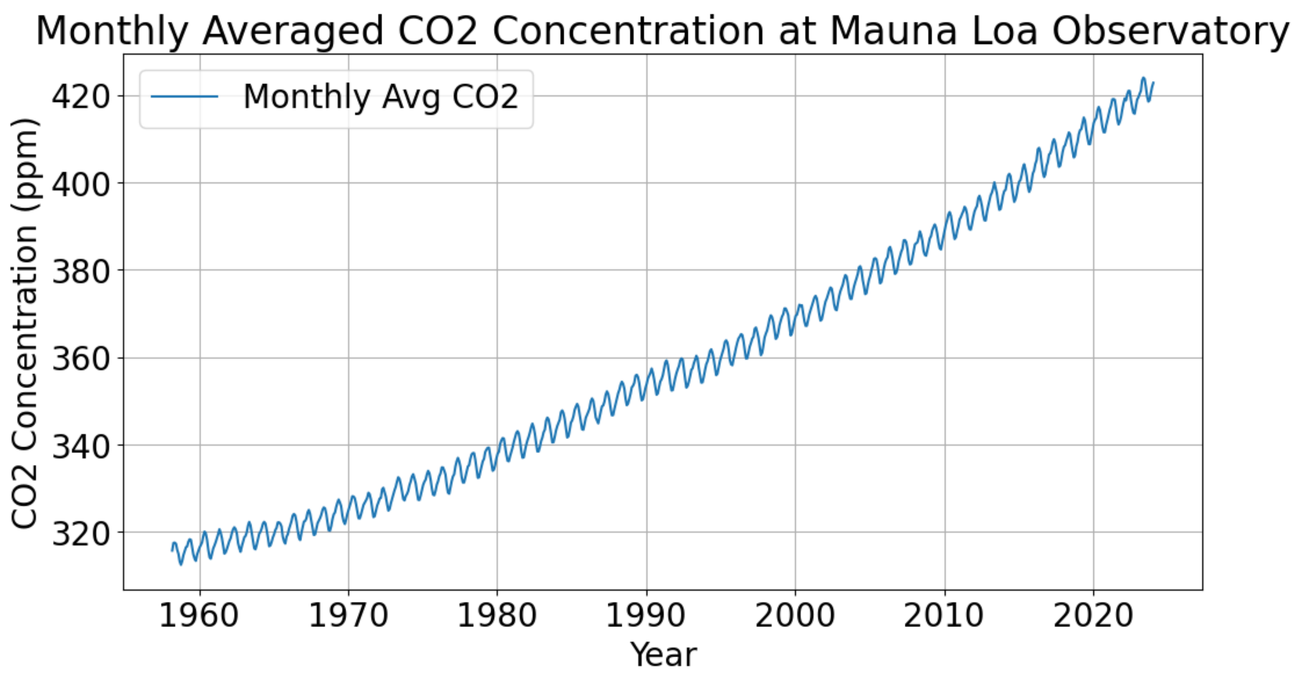
```
tab.describe()
```

Out[5]:

	year	month	decimal date	average	deseasonalized	nday
count	791.000000	791.000000	791.000000	791.000000	791.000000	791.00000
mean	1990.624526	6.505689	1991.124379	358.783704	358.783198	19.0303
std	19.041177	3.452705	19.041344	31.411617	31.362066	12.0096
min	1958.000000	1.000000	1958.202700	312.430000	314.430000	-1.00000
25%	1974.000000	4.000000	1974.666650	330.230000	330.515000	10.50000
50%	1991.000000	7.000000	1991.125000	354.930000	355.260000	25.00000
75%	2007.000000	9.500000	2007.583350	384.280000	384.155000	28.00000
max	2024.000000	12.000000	2024.041700	424.000000	422.560000	31.00000

In [6]:

```
import matplotlib.pyplot as plt

# Plotting the CO2 concentration over time
plt.figure(figsize=(12, 6))
plt.plot(tab['decimal date'], tab['average'], label='Monthly Avg CO2')
plt.xlabel('Year')
plt.ylabel('CO2 Concentration (ppm)')
plt.title('Monthly Averaged CO2 Concentration at Mauna Loa Observatory')
plt.grid(True)
plt.legend()
plt.show()

# Displaying the time span of the dataset
time_span = (tab['decimal date'].min(), tab['decimal date'].max())
time_span
```



```
Out[6]: (1958.2027, 2024.0417)
```

The dataset spans from the year 1958, with a decimal date of 1958.2027, up to the year 2024, with a decimal date of 2024.0417. This provides us with a comprehensive view of over six decades of atmospheric CO2 measurements.

Part 4 Prepare Data

Question 2

(2 pts)

Are there outliers in the dataset? If so remove them and then create your feature (the decimal years) and target arrays (the average CO2 concentration). Use only the data from 1958 to 2010 for training. Make a separate array with the data from the beginning of 2011 to the end of 2019 for testing.

In [7]:
```
# code here

# Checking for outliers in the 'average' by calculating the Z-scores
from scipy.stats import zscore

tab['z_score'] = zscore(tab['average'])

# Considering outliers as those with a z-score greater than 3 or less than -
outliers = tab[(tab['z_score'] > 3) | (tab['z_score'] < -3)]
```

```
# Removing outliers from the dataset
cleaned_tab = tab[(tab['z_score'] <= 3) & (tab['z_score'] >= -3)]

# Preparing the feature (decimal years) and target arrays (average CO2 conce
training_data = cleaned_tab[(cleaned_tab['decimal date'] >= 1958) & (cleanec
X_train = training_data['decimal date'].values
y_train = training_data['average'].values

# Preparing data for testing (beginning of 2011 to the end of 2019)
testing_data = cleaned_tab[(cleaned_tab['decimal date'] >= 2011) & (cleaned_
X_test = testing_data['decimal date'].values
y_test = testing_data['average'].values

# Checking the number of outliers removed
num_outliers_removed = len(outliers)

(num_outliers_removed, X_train.shape, y_train.shape, X_test.shape, y_test.sh
```

```
Out[7]: (0, (634,), (634,), (96,), (96,))
```

Part 5 Select model and train

In examining the model, we can see that there is a long term trend but also some periodic variations. We can use a combination of kernels to model the curve.

Some helpful references for using Gaussian Processes with Scikit Learn are here:

- https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.GaussianProcessRegress
- https://scikit-learn.org/stable/modules/gaussian_process.html

Part 5.1 Practice with Kernels

To understand how to use Gaussian processes, we will practice with start by an example of using kernels in Scikit Learn.

We will start with the Radial Basis Kernel (also known as the squared exponential kernel). This kernel corresponds to a basis set of Gaussians. It is often used to fit time series data that has correlation over some length scale.

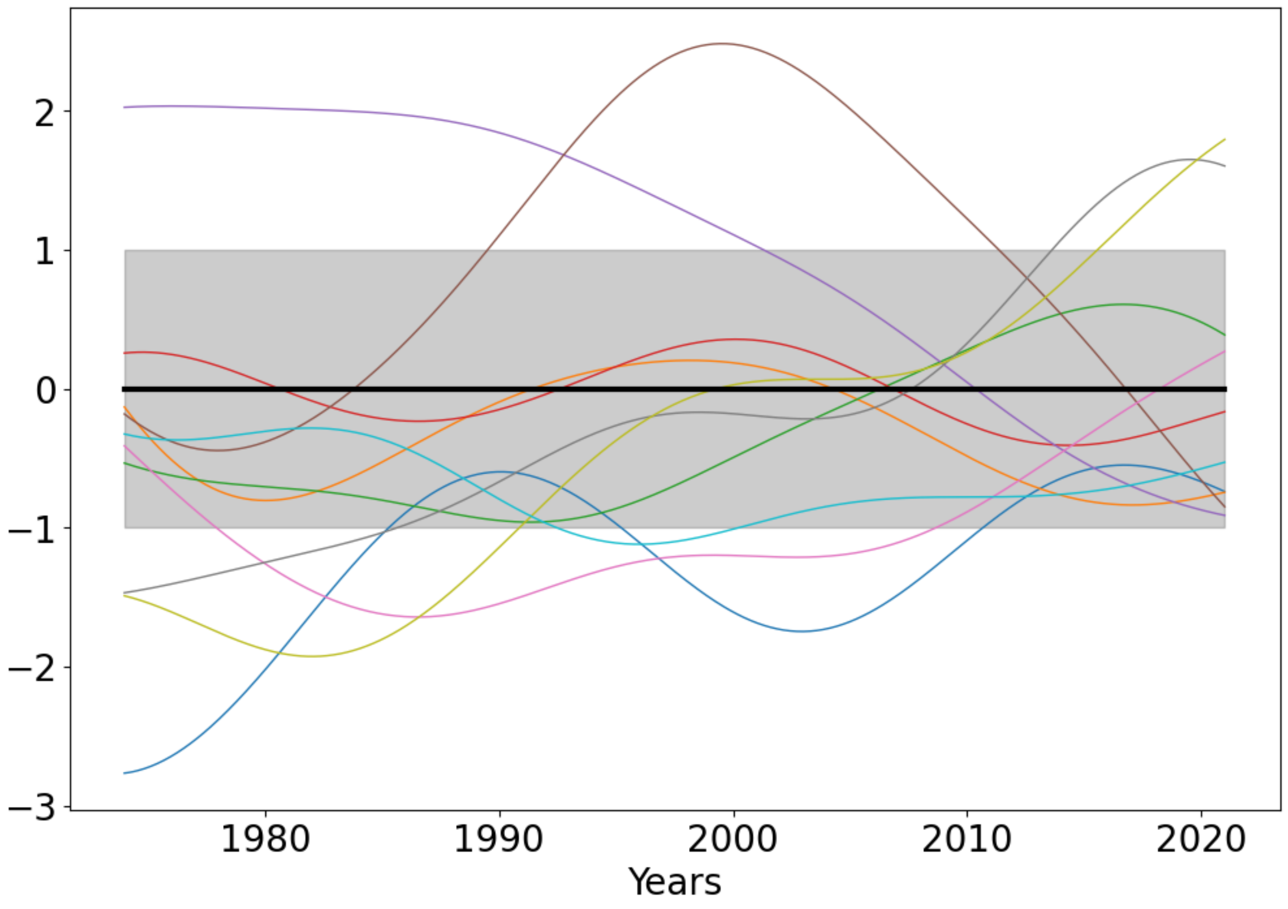
Start by reading about the RBF kernel in Scikit Learn: https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.kernels.RBF.html

Below is some code used to initialize a Gaussian Process regression model and the RBF kernel. Before fitting the data, we can use this model to sample from the prior space of functions that are defined by the kernel. Often it is useful to see the shape and form of the functions that it can model. This will help you get a since of what kernels might be useful for modeling your data.

In [8]:
```
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import (RBF, Matern, RationalQuadratic
                                              ExpSineSquared, DotProduct,
                                              ConstantKernel, WhiteKernel)
```

In [9]:
```
kernel = 1.0 * RBF(length_scale=10.0, length_scale_bounds=(1e-1, 10.0))

gp = GaussianProcessRegressor(kernel=kernel)
plt.figure(figsize=(12,8))
X_ = np.linspace(1974, 2021, 500)
y_mean, y_std = gp.predict(X_[:, np.newaxis], return_std=True)
plt.plot(X_, y_mean, 'k', lw=3, zorder=9)
plt.fill_between(X_, y_mean - y_std, y_mean + y_std,
                 alpha=0.2, color='k')
y_samples = gp.sample_y(X_[:, np.newaxis], 10)
plt.plot(X_, y_samples, lw=1)
plt.xlabel('Years')
```

Out[9]:
```
Text(0.5, 0, 'Years')
```



Question 3

(3 pts)

In the example code above, what are the hyperparameters of the `RBF` kernel? Describe the effect of those parameters.

In [10]:
```
# your answer here
```

The hyperparameters of the RBF kernel include the **length scale** and its **bounds**:

- **Length Scale**: Controls the smoothness of the function. A smaller value results in a more rapidly changing function, while a larger value leads to a smoother function.
- **Length Scale Bounds**: Defines the range within which the length scale can be optimized, allowing the model to adjust the smoothness of the fitted function within these limits.

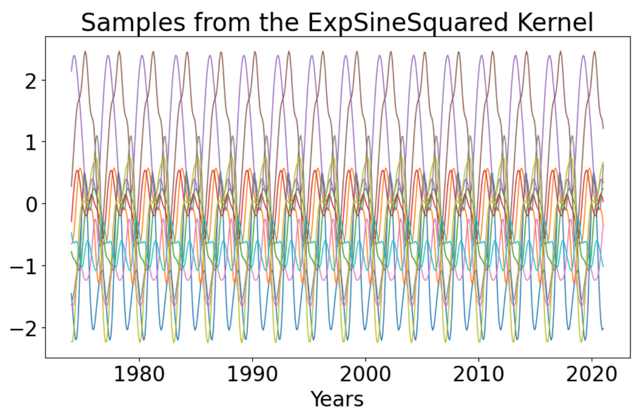
These parameters directly affect the model's ability to capture the correlation and variability in the data.

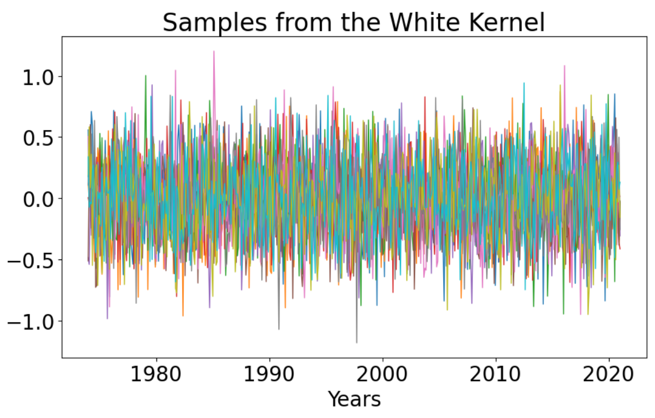
Question 4

(6 pts)

In a similar way as the example above, examine and plot some samples from the `ExpSineSquared` and `WhiteKernel`. What are the hyperparameters of these kernels? Describe qualitatitvely what the samples from the prior look like for these kernels.

In [11]:
```
# code here

# Kernels with their configurations
exp_sine_squared_kernel = ExpSineSquared(length_scale=1.0, periodicity=3.0,
                                         length_scale_bounds=(0.1, 10.0), pe
white_kernel_demo = WhiteKernel(noise_level=0.1, noise_level_bounds=(1e-10,

# Initializing Gaussian Process regressors for each kernel
gp_exp_sine_squared = GaussianProcessRegressor(kernel=exp_sine_squared_kerne
gp_white_demo = GaussianProcessRegressor(kernel=white_kernel_demo)

# Generate sample data
X_ = np.linspace(1974, 2021, 500)[:, np.newaxis]

# Sample from the prior for the ExpSineSquared kernel
y_samples_exp_sine_squared = gp_exp_sine_squared.sample_y(X_, 10)

# Sample from the prior for the White kernel as a learning exercise
y_samples_white_demo = gp_white_demo.sample_y(X_, 10)

# Plotting both ExpSineSquared and White Kernel Samples for comparison
plt.figure(figsize=(18, 6))

# ExpSineSquared Kernel Samples
plt.subplot(1, 2, 1)
plt.plot(X_, y_samples_exp_sine_squared, lw=1)
plt.title('Samples from the ExpSineSquared Kernel')
plt.xlabel('Years')

# WhiteKernel Samples
plt.subplot(1, 2, 2)
plt.plot(X_, y_samples_white_demo, lw=1)
plt.title('Samples from the White Kernel')
plt.xlabel('Years')

plt.tight_layout()
plt.show()
```





ExpSineSquared Kernel: The samples demonstrate periodic functions, showcasing the kernel's ability to capture periodic patterns within data. This is due to its hyperparameters of length scale and periodicity, which control the smoothness of the function and the distance between repetitions, respectively.

White Kernel: The samples from the White Kernel illustrate what appears to be pure noise. This is characteristic of the White Kernel's purpose, which is to model the noise in the data. Its primary hyperparameter, the noise level, determines the variance of this noise. Sampling from a White Kernel thus produces functions that resemble random noise.

Part 5.2 Put together a Gaussian Process Model

One of the great propertiers of Gaussian Processes is that you can simply add kernels together to make more complicated models.

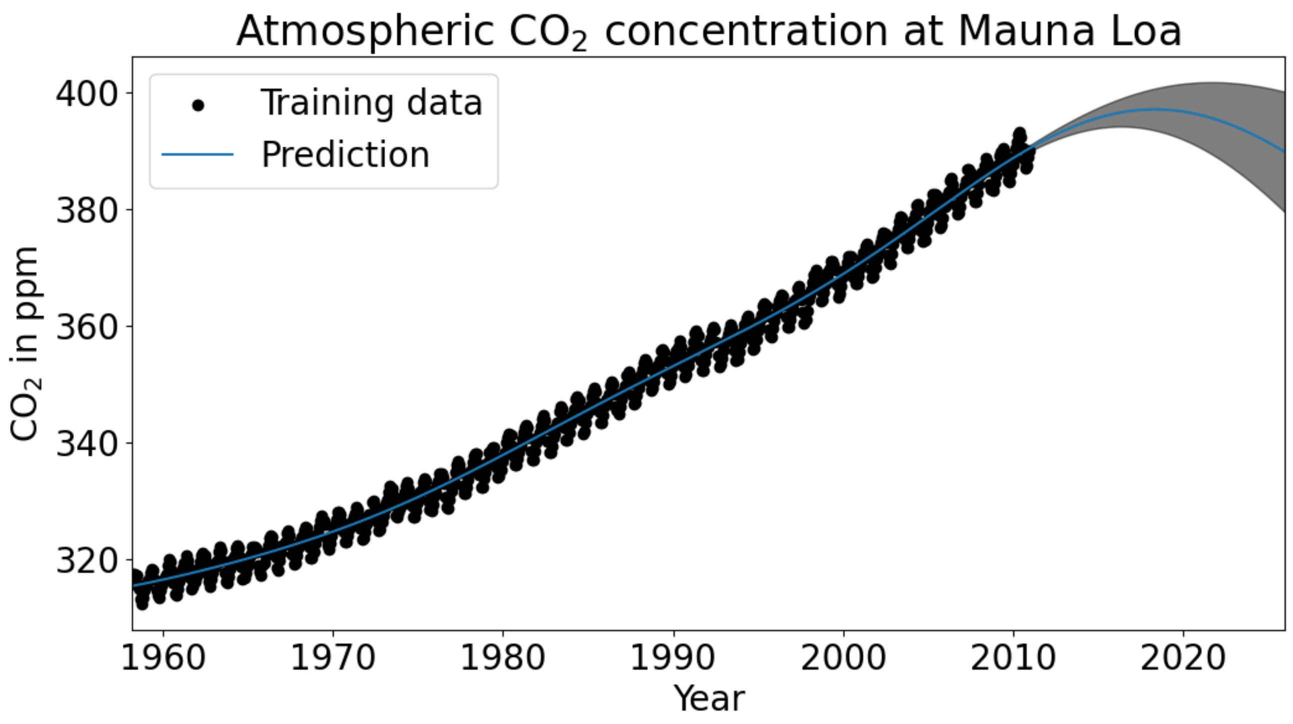
We'll start off first with just the `RBF` kernel to fit the long term trends. Below is some example code to show how to do it with Scikit Learn.

In [12]:
```
# Re-defining X and y for the training data (from 1958 to 2010), reshaping X
X_train_reshape = X_train.reshape(-1, 1)  # Reshaping X_train to be 2D

# Defining the kernel with the provided parameters
k1 = 66.0**2 * RBF(length_scale=67.0)  # long term smooth rising trend
kernel_gpml = k1  # Here we have just 1 kernel

# Initializing and fitting the Gaussian Process Regressor
gp = GaussianProcessRegressor(kernel=kernel_gpml, alpha=0.01, normalize_y=Tr
gp.fit(X_train_reshape, y_train)

# Displaying the fitted kernel and log-marginal-likelihood
fitted_kernel = gp.kernel_
log_marginal_likelihood = gp.log_marginal_likelihood(gp.kernel_.theta)
```

```
(fitted_kernel, log_marginal_likelihood)
```

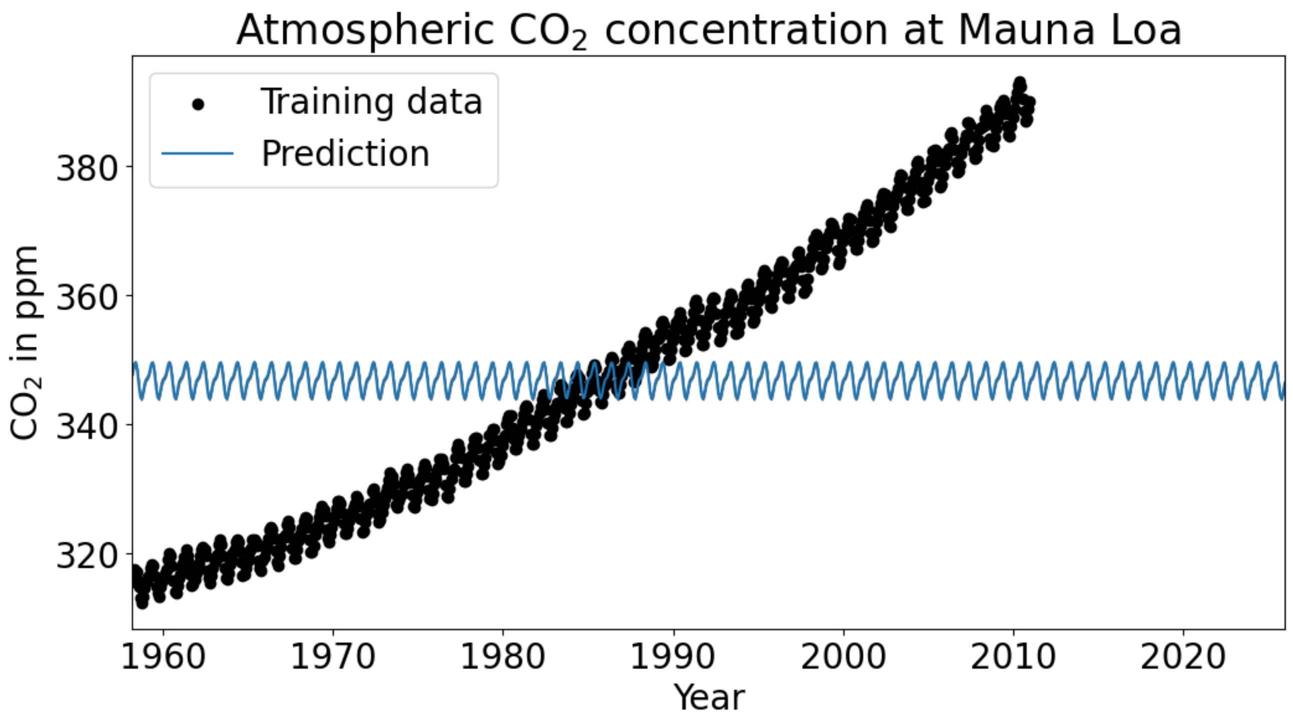
Out[12]:
```
(1.21**2 * RBF(length_scale=16.9), 566.970884170963)
```

In [13]:
```
# Fixing the code to plot predictions, including predictions 15

# Adjusting the linspace to predict 15 years into the future based on the ex
X_future = np.linspace(X_train.min(), X_train.max() + 15, 1000)[:, np.newaxi
y_pred_future, y_std_future = gp.predict(X_future, return_std=True)

# Plotting the predictions along with the training data
plt.figure(figsize=(12, 6))
plt.scatter(X_train, y_train, c='k', label='Training data')
plt.plot(X_future[:, 0], y_pred_future, label='Prediction')
plt.fill_between(X_future[:, 0], y_pred_future - y_std_future, y_pred_future
plt.xlim(X_future.min(), X_future.max())
plt.xlabel("Year")
plt.ylabel(r"CO$_2$ in ppm")
plt.title(r"Atmospheric CO$_2$ concentration at Mauna Loa")
plt.legend()
plt.show()
```



Similarly, we can use instead the periodic kernel `ExpSineSquared` to model the periodic behavoir of the curve.

In [14]:
```
# Defining the ExpSineSquared kernel
k2 = ExpSineSquared(length_scale=1.0, periodicity=1.0, periodicity_bounds="f
kernel_gpml = k2  # Using just the ExpSineSquared kernel

# Initializing and fitting the Gaussian Process Regressor with the ExpSineSq
gp_exp_sine = GaussianProcessRegressor(kernel=kernel_gpml, alpha=0.01, norma
gp_exp_sine.fit(X_train_reshape, y_train)  # Fitting to the training data

# Predicting and plotting for the same period plus 15 years into the future
y_pred_exp_sine, y_std_exp_sine = gp_exp_sine.predict(X_future, return_std=T

# Plotting the predictions
plt.figure(figsize=(12, 6))
plt.scatter(X_train, y_train, c='k', label='Training data')
plt.plot(X_future[:, 0], y_pred_exp_sine, label='Prediction')
plt.fill_between(X_future[:, 0], y_pred_exp_sine - y_std_exp_sine, y_pred_ex
plt.xlim(X_future.min(), X_future.max())
plt.xlabel("Year")
plt.ylabel(r"CO$_2$ in ppm")
plt.title(r"Atmospheric CO$_2$ concentration at Mauna Loa")
plt.legend()
plt.show()

# Displaying the fitted kernel and log-marginal-likelihood
fitted_kernel_exp_sine = gp_exp_sine.kernel_
log_marginal_likelihood_exp_sine = gp_exp_sine.log_marginal_likelihood(gp_ex

(fitted_kernel_exp_sine, log_marginal_likelihood_exp_sine)
```



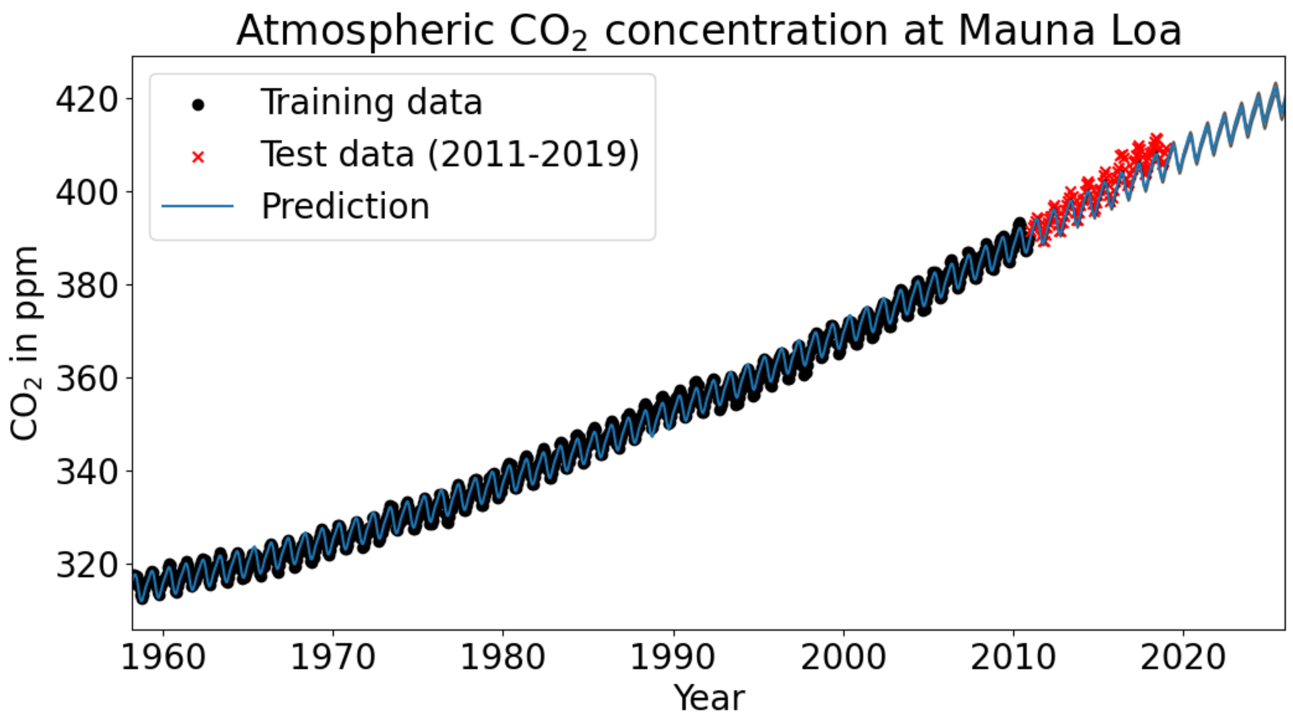
```
Out[14]:  (ExpSineSquared(length_scale=4.43, periodicity=1), -30637.170092044707)
```

Question 5

(4 pts)

Now create a new kernel by adding the `RBF` and the `ExpSineSquared` kernels together. Make a plot of the fit and the prediction. Plot the test data that you saved from 2011 to 2019. Describe the model performance.

In [15]:
```
# code here
# Creating a new kernel by adding the RBF and the ExpSineSquared kernels tog
combined_kernel = 66.0**2 * RBF(length_scale=67.0) + ExpSineSquared(length_s

gp_combined = GaussianProcessRegressor(kernel=combined_kernel, alpha=0.01, r
gp_combined.fit(X_train_reshape, y_train)

y_pred_combined, y_std_combined = gp_combined.predict(X_future, return_std=T

plt.figure(figsize=(12, 6))
plt.scatter(X_train, y_train, c='k', label='Training data')
plt.scatter(X_test, y_test, c='red', label='Test data (2011-2019)', marker='
plt.plot(X_future[:, 0], y_pred_combined, label='Prediction')
plt.fill_between(X_future[:, 0], y_pred_combined - y_std_combined, y_pred_co
plt.xlim(X_future.min(), X_future.max())
plt.xlabel("Year")
plt.ylabel(r"CO$_2$ in ppm")
plt.title(r"Atmospheric CO$_2$ concentration at Mauna Loa")
plt.legend()
plt.show()

fitted_kernel_combined = gp_combined.kernel_
log_marginal_likelihood_combined = gp_combined.log_marginal_likelihood(gp_co

(fitted_kernel_combined, log_marginal_likelihood_combined)
```



```
Out[15]: (6.15**2 * RBF(length_scale=108) + ExpSineSquared(length_scale=5.09, period
         icity=1),
          813.5445013938722)
```

The model now combines the RBF and ExpSineSquared kernels, aiming to capture both the long-term trend and periodic variations in the atmospheric CO_2 concentration data at Mauna Loa. The plot includes predictions extending 15 years into the future from the last year of training data, along with the test data from 2011 to 2019 for comparison.

Model Performance: Fitted Kernel: The optimized kernel is 6.15**2 * RBF(length_scale=108) + ExpSineSquared(length_scale=5.09, periodicity=1). This indicates that the model has adjusted to a variance of approximately 6.15^2 for the RBF component, with a length scale of 108, and an ExpSineSquared component with a length scale of 5.09 and a fixed periodicity of 1 year. Log-Marginal-Likelihood: 813.545. The combined kernal is clearly a much better fit to the data though projections into the future look to underguess so the model is not fully capturing the data.

Part 6 Fine tune model

While the major properties of the CO2 curve was represented by our two kernel model, we can do better by allowing for some variations in the long term trend, variations in the periodic signal, and allowing for noise.

We can do this by adjusting and including more kernels.

Question 6

(6 pts)

Create a kernel the combination of four components:

- The same `RBF` kernel as in Part 5
- Modify the periodic `ExpSineSquared` kernel in Part 5 by **multipyling** it with another `RBF` kernel to allow the periodic signal to vary slightly.
- Add a `RationalQuadratic` kernel to account for medium term irregularies
- Add a `WhiteKernel` to account for white noise (Gaussian noise) in the modeling
- Hint: if you need help with this, see https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpr_co2.html

Fit the data from 1958 to 2010, and the fit the GP model

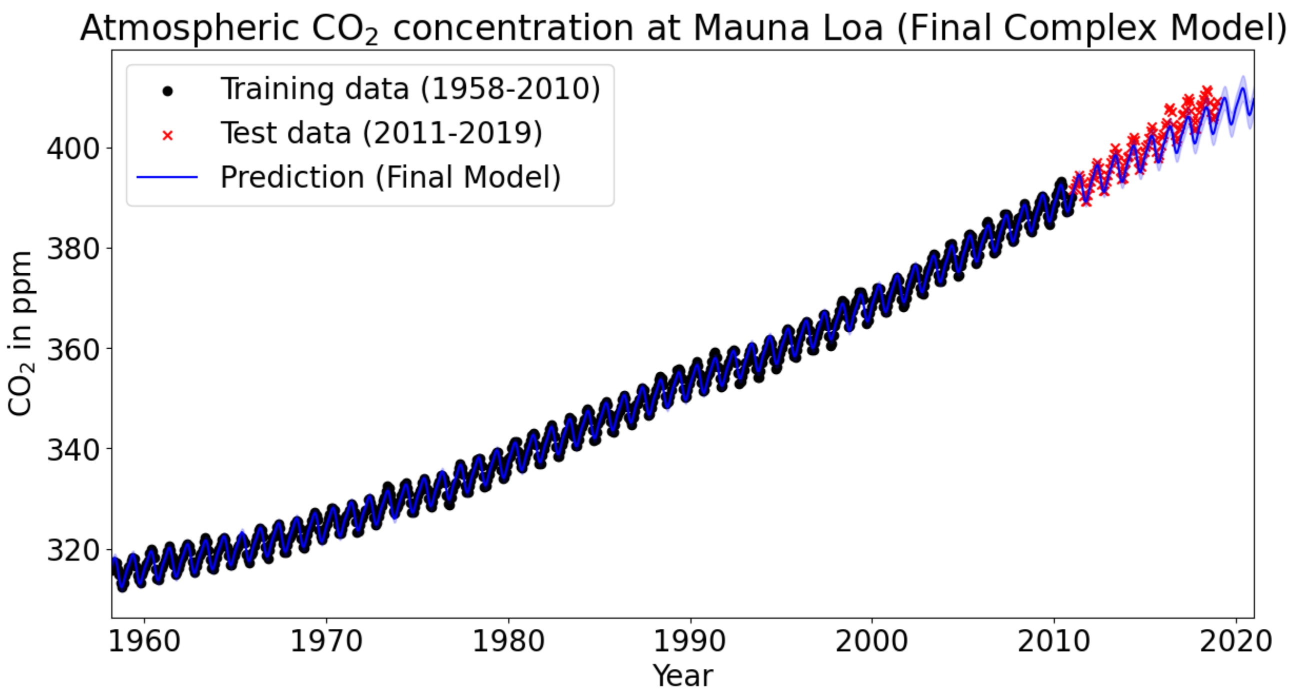
In [16]:
```
# Redefining the complex kernel with an adjusted lower bound for the WhiteKe
final_complex_kernel = (66.0**2 * RBF(length_scale=67.0) +
                        1.0**2 * RBF(length_scale=100.0, length_scale_bounds
                        ExpSineSquared(length_scale=1.0, periodicity=1.0, pe
                        RationalQuadratic(length_scale=1.0, alpha=0.1, lengt
                        WhiteKernel(noise_level=0.5, noise_level_bounds=(1e-

# Re-initializing and fitting the Gaussian Process Regressor with the final
gp_final_complex = GaussianProcessRegressor(kernel=final_complex_kernel, alp
gp_final_complex.fit(X_train_reshape, y_train)

# Displaying the fitted kernel and log-marginal-likelihood for the final com
fitted_kernel_final_complex = gp_final_complex.kernel_
log_marginal_likelihood_final_complex = gp_final_complex.log_marginal_likeli

(fitted_kernel_final_complex, log_marginal_likelihood_final_complex)
```

```
/opt/conda/lib/python3.11/site-packages/sklearn/gaussian_process/kernels.py:
419: ConvergenceWarning: The optimal value found for dimension 0 of paramete
r k2__noise_level is close to the specified lower bound 0.001. Decreasing th
e bound and calling fit again may find a better value.
  warnings.warn(
```

Out[16]:
```
(3.93**2 * RBF(length_scale=76.1) + 0.266**2 * RBF(length_scale=9.68e+03) *
ExpSineSquared(length_scale=2.51, periodicity=1) + RationalQuadratic(alpha=
0.00258, length_scale=134) + WhiteKernel(noise_level=0.001),
 798.7754773915142)
```

Plot the data and the predictions. Plot the 2011-2021 data and discuss how well it fits. Compare the prediction to that from Part 5.

In [17]:
```
# code here

# Preparing data for plotting predictions and comparing with actual data fro
X_test_reshape = X_test.reshape(-1, 1)  # Reshaping the test data for predic

# Predicting CO2 concentration from 1958 to 2021 using the final complex moc
X_extended = np.linspace(X_train.min(), 2021, 1000)[:, np.newaxis]  # Extenc
y_pred_final, y_std_final = gp_final_complex.predict(X_extended, return_std=

# Plotting
plt.figure(figsize=(14, 7))
plt.scatter(X_train, y_train, c='k', label='Training data (1958-2010)')
plt.scatter(X_test, y_test, c='red', label='Test data (2011-2019)', marker='
plt.plot(X_extended[:, 0], y_pred_final, 'b', label='Prediction (Final Model
plt.fill_between(X_extended[:, 0], y_pred_final - y_std_final, y_pred_final
plt.xlim(X_extended.min(), X_extended.max())
plt.xlabel("Year")
plt.ylabel(r"CO$_2$ in ppm")
plt.title(r"Atmospheric CO$_2$ concentration at Mauna Loa (Final Complex Moc
plt.legend()
plt.show()
```



Compared to the model in Part 5, which used a simpler kernel configuration, this final complex model—with its combination of RBF, ExpSineSquared, RationalQuadratic, and WhiteKernel components—offers a nuanced approach to modeling the CO_2 data.

We see a strong fit to the data overall but still see that we begin to underestimate C02 ppm after 2011 and the deviation increases with time.

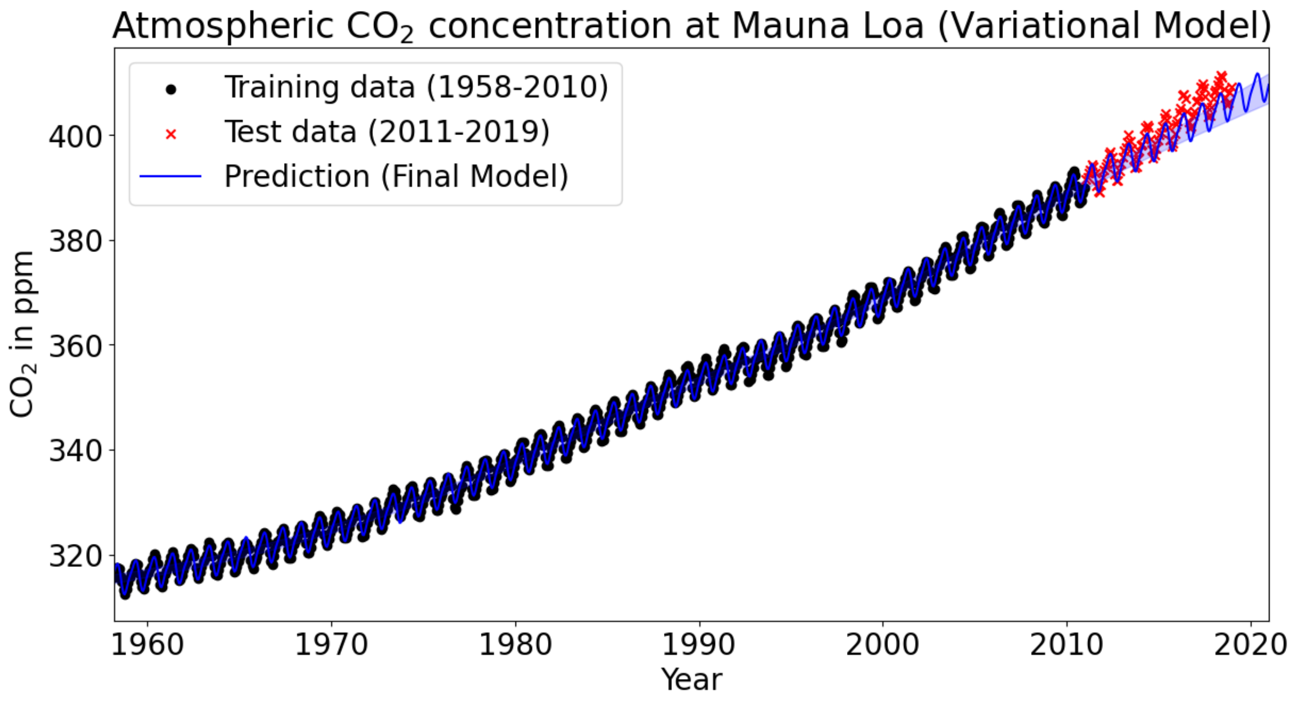
Question 7

(3 pts)

Explore some variations in the kernels to see if you can get a better prediction

In [18]:
```
# code here

# Exploring variations in the kernels to potentially improve the prediction
# Adjusting the kernel configuration to include different parameters and com

# Variation 1: Adjusting the length scales and combining kernels differently
variation_kernel_1 = (66.0**2 * RBF(length_scale=67.0) +
                      1.0**2 * RBF(length_scale=90.0, length_scale_bounds=
                      ExpSineSquared(length_scale=1.1, periodicity=0.9, pe
                      RationalQuadratic(length_scale=1.0, alpha=0.1, lengt
                      WhiteKernel(noise_level=0.5, noise_level_bounds=(1e-

# Initializing and fitting the Gaussian Process Regressor with the variation
gp_variation_1 = GaussianProcessRegressor(kernel=variation_kernel_1, alpha=0
gp_variation_1.fit(X_train_reshape, y_train)

y_pred_variation_1, y_std_variation_1 = gp_variation_1.predict(X_extended, r

# Plotting the predictions for the variation model
plt.figure(figsize=(14, 7))
plt.scatter(X_train, y_train, c='k', label='Training data (1958-2010)')
plt.scatter(X_test, y_test, c='red', label='Test data (2011-2019)', marker='
plt.plot(X_extended[:, 0], y_pred_final, 'b', label='Prediction (Final Model
plt.fill_between(X_extended[:, 0], y_pred_variation_1 - y_std_variation_1, y
plt.xlim(X_extended.min(), X_extended.max())
plt.xlabel("Year")
plt.ylabel(r"CO$_2$ in ppm")
plt.title(r"Atmospheric CO$_2$ concentration at Mauna Loa (Variational Model
plt.legend()
plt.show()

# Displaying the fitted kernel and log-marginal-likelihood for the variation
fitted_kernel_variation_1 = gp_variation_1.kernel_
log_marginal_likelihood_variation_1 = gp_variation_1.log_marginal_likelihood

(fitted_kernel_variation_1, log_marginal_likelihood_variation_1)
```

```
/opt/conda/lib/python3.11/site-packages/sklearn/gaussian_process/kernels.py:
419: ConvergenceWarning: The optimal value found for dimension 0 of paramete
r k2__noise_level is close to the specified lower bound 0.001. Decreasing th
e bound and calling fit again may find a better value.
  warnings.warn(
```



```
Out[18]: (3.54**2 * RBF(length_scale=71.1) + 0.21**2 * RBF(length_scale=591) * ExpSi
         neSquared(length_scale=1.3e+03, periodicity=0.9) + RationalQuadratic(alpha=
         0.00265, length_scale=131) + WhiteKernel(noise_level=0.001),
          566.917054051137)
```

Assessing the effects of COVID19 on CO2 levels

Question 8

(2 pts)

First, without looking at the data, predict would happen to the CO2 levels in 2020 to recent times. Could the COVID19 pandemic affect these levels? Why or why not?

The COVID-19 pandemic likely led to a temporary reduction in CO2 emissions due to decreased industrial activity and travel. However, the overall impact on atmospheric CO2 levels would be minimal because CO2 accumulates over long periods, and the atmosphere's response to short-term emissions reductions is slow. Any immediate changes would probably be small and possibly hard to detect against the backdrop of natural variability.

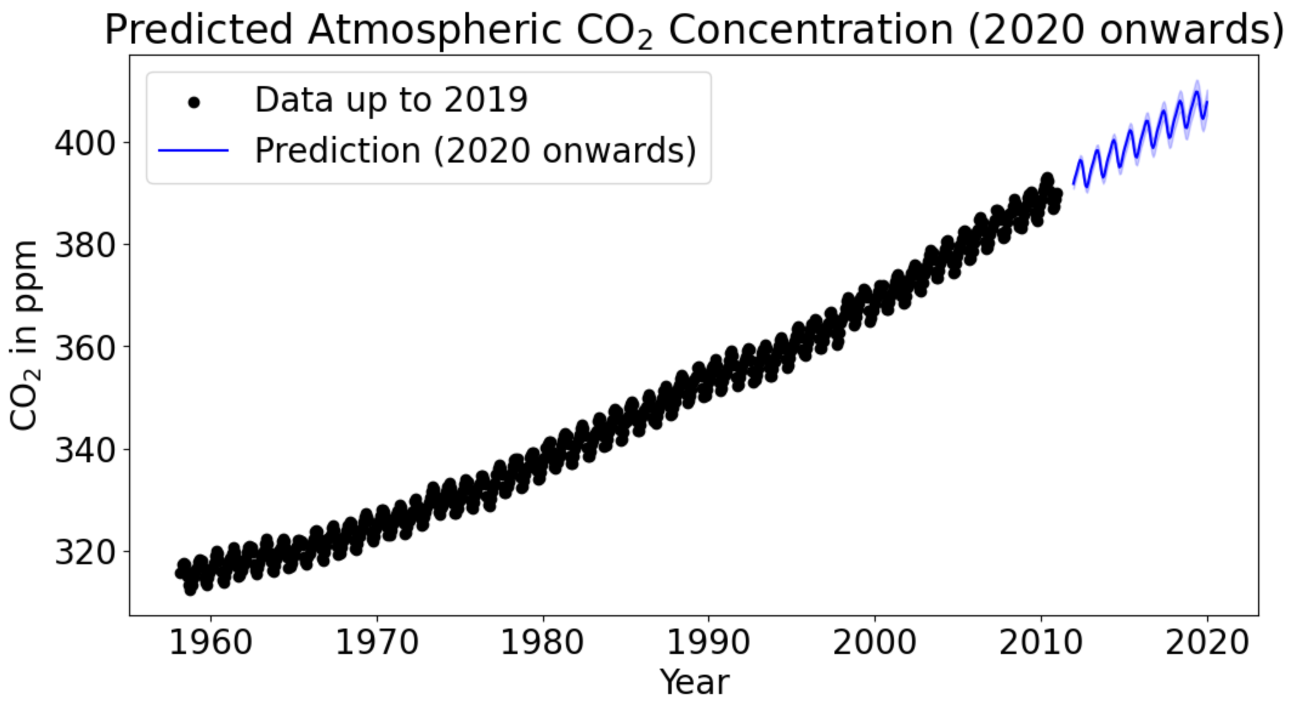
Question 9

(2 pts)

Now, go back and fit your GP model on data from up to 2019. Then, predict the CO2 levels in 2020 to the most recent year

In [19]:
```
# Adjusting the dataset to fit the GP model on data up to 2019
X_train_up_to_2019 = X_train[X_train < 2020]
y_train_up_to_2019 = y_train[X_train < 2020]
X_train_up_to_2019_reshape = X_train_up_to_2019.reshape(-1, 1)  # Reshaping

# Re-fitting the Gaussian Process Regressor with the previously defined fina
gp_final_for_prediction = GaussianProcessRegressor(kernel=final_complex_kerr
gp_final_for_prediction.fit(X_train_up_to_2019_reshape, y_train_up_to_2019)

# Predicting CO2 levels from 2020 to the most recent year in the dataset
X_predict_from_2020 = np.linspace(2020, X_train.max() + 1, 100)[:, np.newaxi
y_pred_from_2020, y_std_from_2020 = gp_final_for_prediction.predict(X_predic

# Plotting the prediction for 2020 onwards
plt.figure(figsize=(12, 6))
plt.scatter(X_train_up_to_2019, y_train_up_to_2019, c='k', label='Data up tc
plt.plot(X_predict_from_2020[:, 0], y_pred_from_2020, 'b', label='Predictior
plt.fill_between(X_predict_from_2020[:, 0], y_pred_from_2020 - y_std_from_2C
plt.xlabel("Year")
plt.ylabel(r"CO$_2$ in ppm")
plt.title(r"Predicted Atmospheric CO$_2$ Concentration (2020 onwards)")
plt.legend()
plt.show()
```

```
/opt/conda/lib/python3.11/site-packages/sklearn/gaussian_process/kernels.py:
419: ConvergenceWarning: The optimal value found for dimension 0 of paramete
r k2__noise_level is close to the specified lower bound 0.001. Decreasing th
e bound and calling fit again may find a better value.
  warnings.warn(
```



Question 10

(3 pts)

Describe whether what you found meets your expectations. Why or why not?

I believe the findings meet expectations. We found a combined kernel to model the data and used it as a seed to find for for the years up to 2019. Based on the dataset itself it does not appear that the COVID-19 pandemic had a very noticable impact on C02 ppm as measured.

Part 7 Present solution

Question 11

(4 pts)

Discuss the model which appears to perform best: what kernels did you end up using? What are the hyperparameters of the best kernel?

```
In [20]: # your answer here
```

The futrure predictions continue to model the seasonality of C02 concentrations but begin to deviate significanly from the overall trend in the data after a few years. The overall trend of the data is consistent but the exact rate of change varries over years and decades making it difficult to model the far future with confidence.

The best preforming model was the 'final complex model'. It used a combined kernel, employing, RBF, EXPSineSquared, RationalQuadratic, and White Noise in order to attempt to capture the overall trends, periodicity, and psuedorandomness in the data. The hyperparameters of best fit are listed below.

In [21]:
```
# Final Complex Model hyperparmeters for reference
final_complex_kernel = (66.0**2 * RBF(length_scale=67.0) +
                        1.0**2 * RBF(length_scale=100.0, length_scale_bounds
                        ExpSineSquared(length_scale=1.0, periodicity=1.0, pe
                        RationalQuadratic(length_scale=1.0, alpha=0.1, lengt
                        WhiteKernel(noise_level=0.5, noise_level_bounds=(1e-
```

Part 8 Launch, Monitor and Maintain

Nothing to here for this lab.

All done! Nice job!

In [ ]:
```
```