Analysis of the Time Series Data in Electricity and Gas Production Across the United States

Navneet Rajagopal

nrajagopal@ucsb.edu

Abstract

This project aims to forecast and predict electricity projections using Autoregressive Integrated Moving Average (ARIMA) and Support Vector Regression (SVR) models. The analysis utilizes historical data on industrial production of electric and gas utilities in the United States from 1985 to 2018, excluding territories. By combining these models, the project seeks to provide accurate insights into electricity trends, aiding in resource allocation and decision-making in the energy sector.

Introduction

Accurate forecasting of electricity production is crucial for effective energy management, resource allocation, and policy planning in the power sector. This project focuses on utilizing time series analysis to forecast and predict electricity projections. The analysis employs two different models: Autoregressive Integrated Moving Average (ARIMA) and Support Vector Regression (SVR). These models will be applied to historical data on industrial production of electric and gas utilities in the United States, subset from 1985 to 2018. The dataset represents the real output of all relevant establishments in the United States, excluding territories. Access to the dataset can be found through the provided link.

The significance of forecasting electricity production lies in its wide-ranging implications for various stakeholders. Energy companies can utilize accurate projections to optimize production schedules, determine infrastructure requirements, and plan for future energy demand. As a shift towards renewable energy becomes eminent, this data will help companies understand what scale they need to produce at. Regulators and policymakers can make informed decisions based on reliable forecasts, aiding in the development of renewable energy policies and initiatives.

By applying the ARIMA, and SVR models to the industrial production dataset, this project aims to provide valuable insights into electricity trends and future projections. The findings and analysis from this study can contribute to the existing body of research on energy forecasting and assist in better understanding and managing electricity production dynamics.

Data:

DATE	Value
01-01-1985	72.5052
02-01-1985	70.672
03-01-1985	62.4502
04-01-1985	57.4714
05-01-1985	55.3151
06-01-1985	58.0904
07-01-1985	62.6202
08-01-1985	63.2485
09-01-1985	60.5846
10-01-1985	56.3154
11-01-1985	58.0005
12-01-1985	68.7145
01-01-1986	73.3057
02-01-1986	67.9869
03-01-1986	62.2221
04-01-1986	57.0329
05-01-1986	55.8137
06-01-1986	59.9005
07-01-1986	65.7655

The dataset used in this project encompasses historical data on industrial production of electric and gas utilities in the United States, covering the period from 1985 to 2018. The data represents the real output of relevant establishments in the United States, excluding territories. It provides a comprehensive measurement of the electricity production of various establishments, regardless of ownership.

The reason I chose to use this dataset is simple, everything around us revolves around gas and electricity. The computer I am typing this report on runs on electricity. The air conditioning that I am currently enjoying also comes from electricity. The hot water I use to take a shower comes from gas. All the things that modern society has gotten accustomed to is a result of the rise of electricity and gas. I want to see how as society has grown and developed, the usage of electricity and gas and how much more we have had to produce to come from that.

This data was collected by the Federal Reserve of Saint Louis who collected all the data from a lot of days from 1985 onwards. This data is now publicly available and many states and cities have reported their production numbers so it can be web scrapped by other users as well.

The goal of this dataset is to witness the trends of electricity production, which will almost certainly have some level of seasonality (summer uses more electricity and winter uses more gas), and analyze the common things that happen.

Methodology

For this project I used ARIMA and Support Vector Regression (SVR) models to analyze and forecast the historical data of industrial production of electric and gas utilities in the United States.

The data manipulation involved transforming the data to resemble a stationary data set and declaring the Date column as datetime. This transformation was achieved by using a log transformation and then differencing, eliminating the need for a SARIMA model.

ARIMA (Autoregressive Integrated Moving Average) models was used in this project due to common use in time series analysis. ARIMA models are particularly useful for datasets that demonstrate autocorrelation, such as the one used in this project. By inspecting the ACF and PACF plots of the data and using a grid search we found the optimal order of analysis..

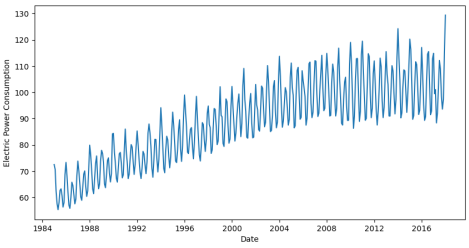
The SVR (Support Vector Regression) model, a type of Support Vector Machine, was used due to its effectiveness in handling high-dimensional data and providing flexibility in model specification. SVR attempts to fit the best line within a specified or predefined margin.

Model estimation and diagnostic techniques were used to analyze electricity and gas production. These techniques included ACF and PACF plots, AIC, Dicky-Fuller, grid searches etc. The information collected from these diagnostics was crucial for the selection of the optimal order of lags and degree of differencing, which directly influences the accuracy of the forecasts.

Finally, the analysis and diagnostic tools used in the estimation process were utilized to generate predictions of electricity and gas production. These predictions provide insights into the future trends of electricity and gas usage.

Results

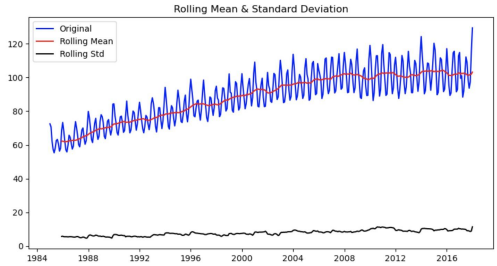
ARIMA:



Appendix 2

First we plot the time series data to see if there needs to be any change. We want to visualize a seasonality and make sure the data isn't too much of a trend in the positive or negative direction. Looking at our data there appears to be seasonality (it goes up and down throughout the years) and has a slight trend (non stationary) that we can normalize with a transformation. Then I ran an augmented dicky fuller test and acf/pcf test to see if the data is stationary (although we can see it is not). Since the p value is greater than 0.05. It is clear that we need to remove the trend and make the data stationary.

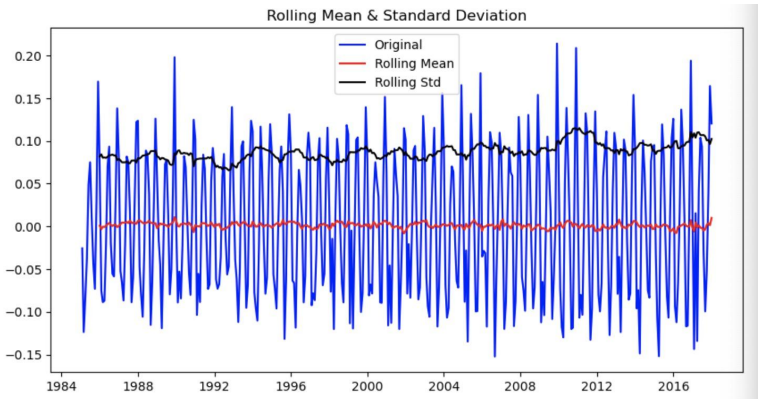
Appendix 3




```
Results of Dickey-Fuller Test:
Test Statistic                  -2.256990
p-value                          0.186215
#Lags Used                      15.000000
Number of Observations Used    381.000000
Critical Value (1%)             -3.447631
Critical Value (5%)             -2.869156
Critical Value (10%)            -2.570827
dtype: float64
```

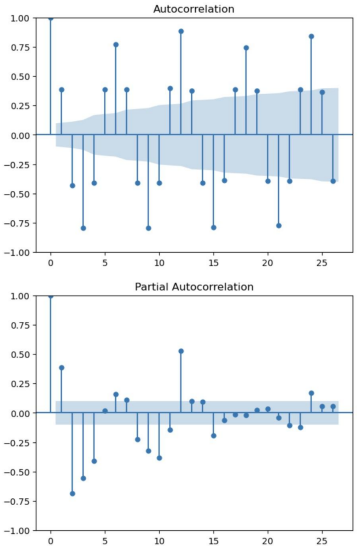
Now I run the log transformation on the data. The reason I do this is to make the data resemble a stationary dataset with no trends. I will be taking the first difference of the logs to get the values that we want in the time series that properly represents it. As we see here the data is far more stationary. We have a dicky-fuller value that is far less that 0.05 which means that the data is stationary. When we look at it there is also not much stationary which is why we can use an ARIMA model to predict the dataset.

Appendix 6




```
Results of Dickey-Fuller Test:
Test Statistic                -6.748333e+00
p-value                        2.995161e-09
#Lags Used                     1.400000e+01
Number of Observations Used    3.810000e+02
Critical Value (1%)           -3.447631e+00
Critical Value (5%)           -2.869156e+00
Critical Value (10%)          -2.570827e+00
dtype: float64
```

Appendix 7

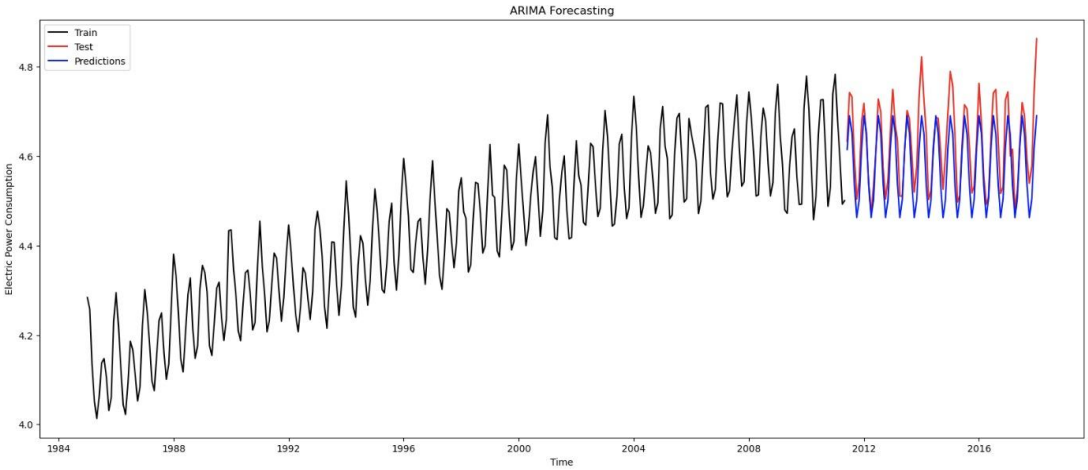


Before we can predict the dataset we need to print the ACF and PACF to find out what order is best for our prediction. But I found the ACF and PACF rather convoluted in their depiction so I decided to use a grid search. This method will fit the ARIMA model to your data for each combination of parameters within a defined range, then it will select the combination that results in the model with the lowest AIC or BIC value. From here I found the best order is 4,1,4.

When we fit the ARIMA, we are separated into a training and testing set, where we train the data and then fit the ARIMA. The ARIMA specifically is using the log transformed data because of the assumptions we have to make. We can then take the predictions that this model will make about the future and plot it. As seen it is relatively close to the testing data and when we run the diagnostics of MSE, MAE, and RMS all the values are really small. That means that this value is good at making predictions about the future.

The general trend of this dataset seems to be good and we can use the ARIMA model to make predictions about the future.

Appendix 9 & 10



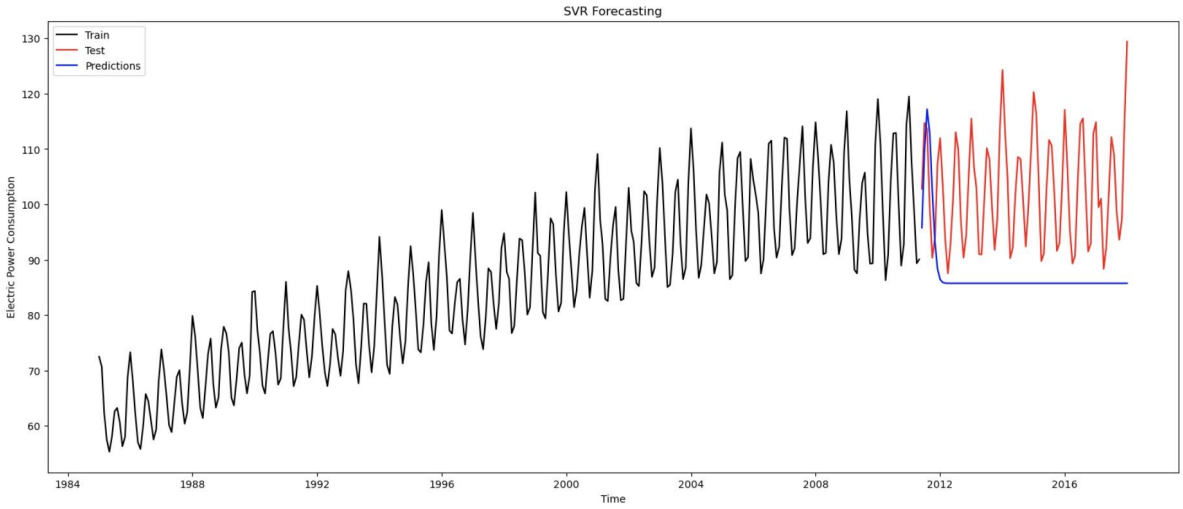

```
Mean Absolute Error: 0.0510744667924506 25
Mean Squared Error: 0.003904609438180405
Root Mean Squared Error: 0.06248687412713493
```

SVR

The SVR model is relatively similar in comparison. There are far less checks we need to do when compared to the ARIMA model because we have already completed them. Since we have already split the data and done everything to reduce stationary and seasonality we can just use the previous split.

An interesting thing about SVR is that it requires 2D inputs and our array is currently a 1D array, so after we do that we can run the SVR model and fit it. Similar to the ARIMA we can find the predictions it makes and then display it.

Appendix 11



In comparison to the ARIMA model this SVR model is significantly worse. The problem here is the model starts well before completely collapsing. This could be a limitation of the indexed data. There could be some over fitting present which has caused the data to collapse, or even the lack of station. But I think the main reason the SVR model is doing much worse is because the regression cannot be properly computed with a dataset that goes up and down. I do not think it is able to properly fit an equation to that with the levels of ups and downs present in this data set. It can be just because SVR is a regression model and it is trying to find a line that will best fit what is seen.

Appendix 12

```
Mean Absolute Error: 0.1872063346843033 7
Mean Squared Error: 0.04389069671047716
Root Mean Squared Error: 0.20950106613207764
```

If we look at the errors, they aren't that big. However it is still significantly lower than the values of the ARIMA model, which is our best fit. Not to mention, the graph looks skewed compared to an ARIMA which gives us another reason to not use this as our best forecast model.

Conclusions

Given the complexity of forecasting electricity production no model can be deemed as perfect. Instead we want to use a combination of models and understand which is the best in a given scenario. In this project we got to use two models to forecast the future of electricity production. We got to understand the weaknesses in some models and the strengths in others that ultimately gave us a good understanding of what we can use for the future of forecasting electricity production.

For example, while the SVR model held low errors in calculations, the graph of the prediction was quite abhorrent. There was a sudden drop in the prediction and it was unable to predict after a little bit. This is no fault of the code or the model, but rathera limitation in the models ability to predict data that fluctuates. It is a regression which implies its inability to handle something like that. On the other hand, the ARIMA model held a great prediction. It had low errors and strong predictions on the graph. The model is suited for a time series like this where there is both an autoregressive and moving average trait in the graph. The ARIMA model is a great use in the future for an investor or official who wants to analyze the trends of electricity production. There are many uses of this sort of information in both the private and public sector.

In terms of improvement, there are several other models we could have ran on this data (GARCH, Boosted Gradient, Long Memory, etc). These are all possibilities of running in the future. The way I see it, there is a lot of potential in the electricity sector, as technology continues to grow and the push towards renewable energy continues. If in the future, someone wants to use a model to predict the Electricity production of the United States, I recommend starting with an ARIMA model.

Harris, Charles R., K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, et al. 2020. “Array Programming with NumPy.” *Nature* 585 (7825): 357–62. https://doi.org/10.1038/s41586-020-2649-2.

Hunter, J. D. 2007. “Matplotlib: A 2D Graphics Environment.” *Computing in Science & Engineering* 9 (3): 90–95. https://doi.org/10.1109/MCSE.2007.55.

“Pandas-Dev/Pandas: Pandas.” 2020. Zenodo. https://doi.org/10.5281/zenodo.3509134.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, et al. 2011. “Scikit-Learn: Machine Learning in Python.” *Journal of Machine Learning Research* 12: 2825–30.

Seabold, Skipper, and Josef Perktold. 2010. “Statsmodels: Econometric and Statistical Modeling with Python.” In *9th Python in Science Conference*.

Waskom, Michael L. 2021. “Seaborn: Statistical Data Visualization.” *Journal of Open Source Software* 6 (60): 3021. https://doi.org/10.21105/joss.03021.

In [27]:
```
import warnings
from datetime import datetime
from math import sqrt
import itertools
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.stattools import adfuller, acf, pacf
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
%matplotlib inline
warnings.filterwarnings('ignore')
```

Appendix 1

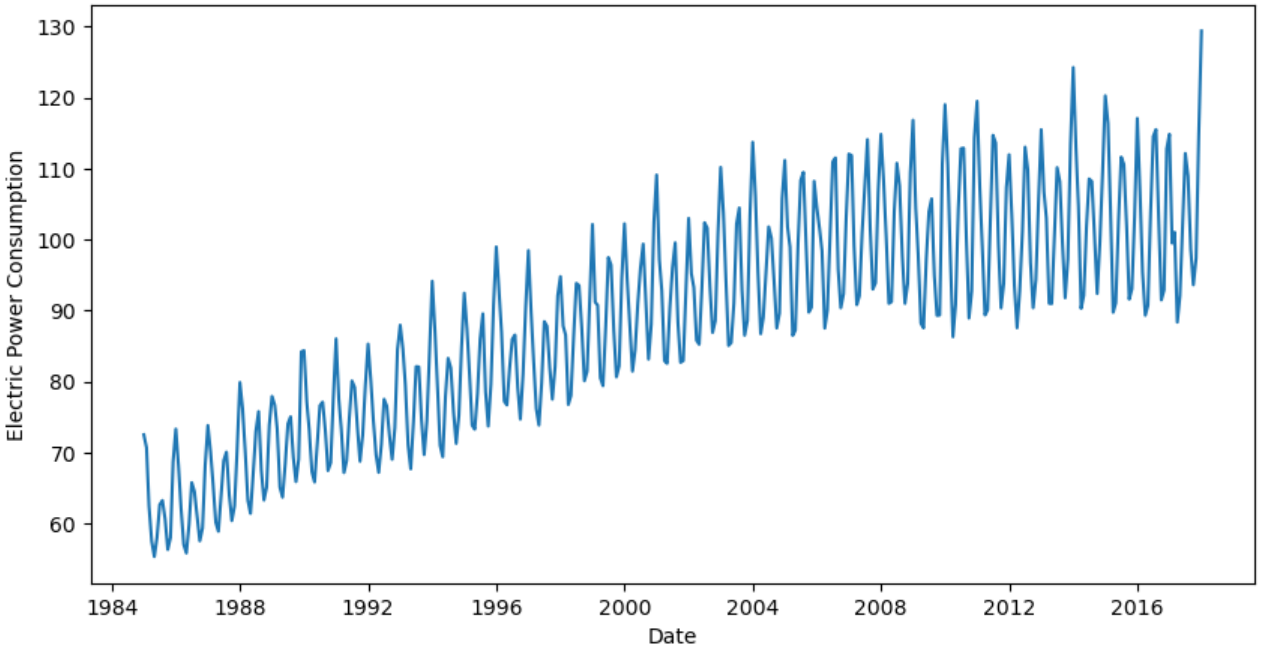
In [28]:
```
# Fix data
df = pd.read_csv('data/Electric_Production.csv')
df['DATE'] = pd.to_datetime(df['DATE'], infer_datetime_format=True)
df_index = df.set_index('DATE', inplace=False)

# Print
print(df_index)
```

```
               Value
DATE
1985-01-01   72.5052
1985-02-01   70.6720
1985-03-01   62.4502
1985-04-01   57.4714
1985-05-01   55.3151
...              ...
2017-09-01   98.6154
2017-10-01   93.6137
2017-11-01   97.3359
2017-12-01  114.7212
2018-01-01  129.4048

[397 rows x 1 columns]
```

Appendix 2

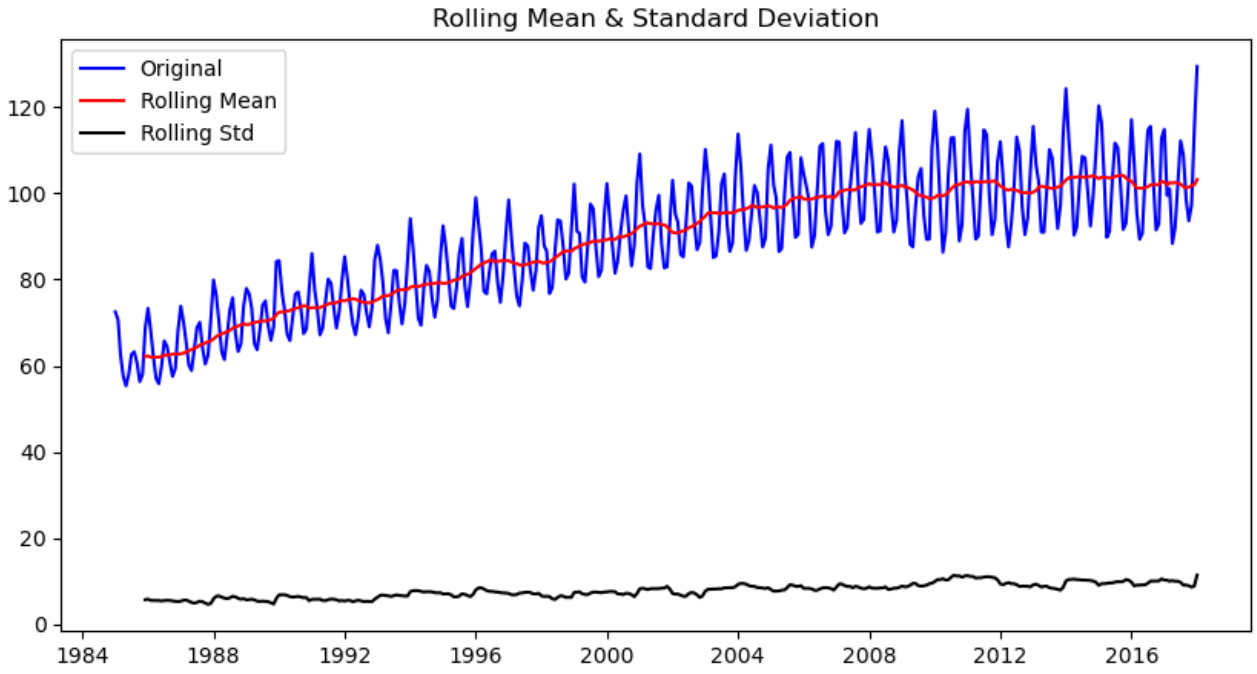
In [29]:
```
#plot
plt.figure(figsize=(10,5))
plt.xlabel('Date')
plt.ylabel('Electric Power Consumption')
plt.plot(df_index)
```

Out[29]:
```
[<matplotlib.lines.Line2D at 0x7ffb40467430>]
```



Appendix 3

In [30]:
```
def get_rolling_statistics(timeseries):
    return timeseries.rolling(window=12).mean(), timeseries.rolling(window=1

def plot_statistics(timeseries, rolling_mean, rolling_std):
    plt.figure(figsize=(10, 5))
    plt.plot(timeseries, color='blue', label='Original')
    plt.plot(rolling_mean, color='red', label='Rolling Mean')
    plt.plot(rolling_std, color='black', label='Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show()

def perform_dickey_fuller_test(timeseries):
    print('Results of Dickey-Fuller Test:')
    test = adfuller(timeseries['Value'], autolag='AIC')
    results = pd.Series(test[0:4], index=['Test Statistic', 'p-value', '#Lag
    for key, value in test[4].items():
        results['Critical Value (%s)' % key] = value
    print(results)

def analyze_stationarity(timeseries):
    rolling_mean, rolling_std = get_rolling_statistics(timeseries)
    plot_statistics(timeseries, rolling_mean, rolling_std)
    perform_dickey_fuller_test(timeseries)

analyze_stationarity(df_index)
```

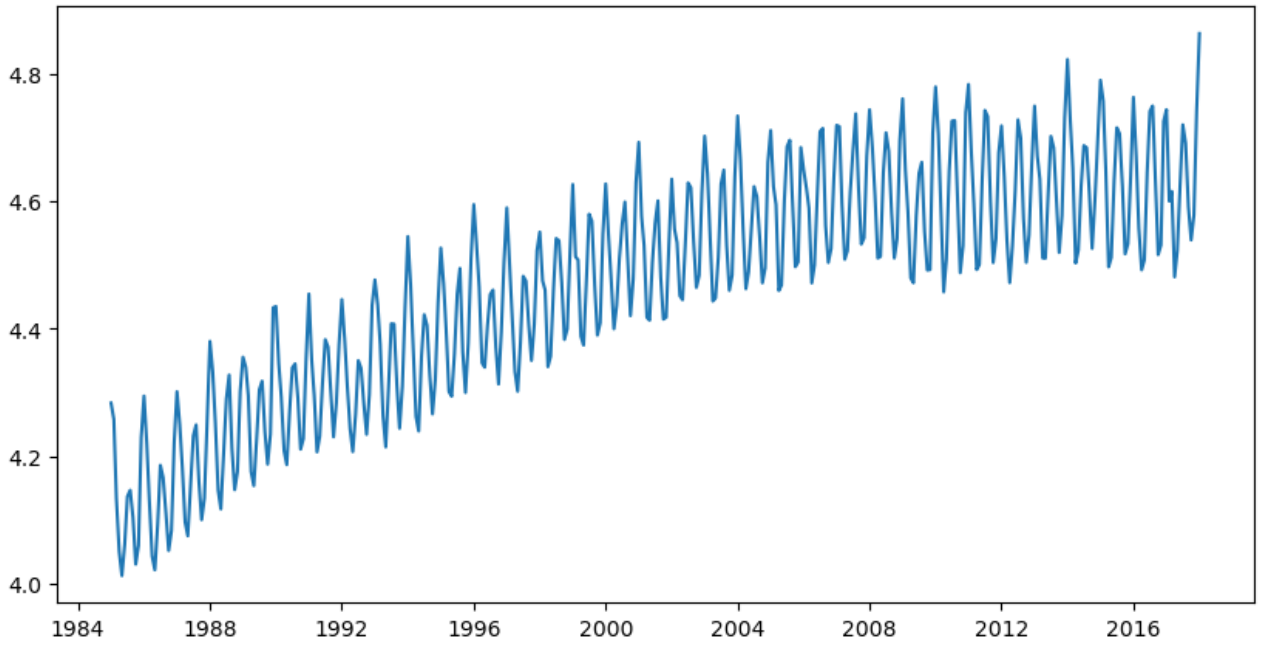


```
Results of Dickey-Fuller Test:
Test Statistic                  -2.256990
p-value                          0.186215
#Lags Used                      15.000000
Number of Observations Used    381.000000
Critical Value (1%)             -3.447631
Critical Value (5%)             -2.869156
Critical Value (10%)            -2.570827
dtype: float64
```

Appendix 4

In [31]:
```
#log transform
df_log = np.log(df_index)
plt.figure(figsize=(10,5))
plt.plot(df_log)
```

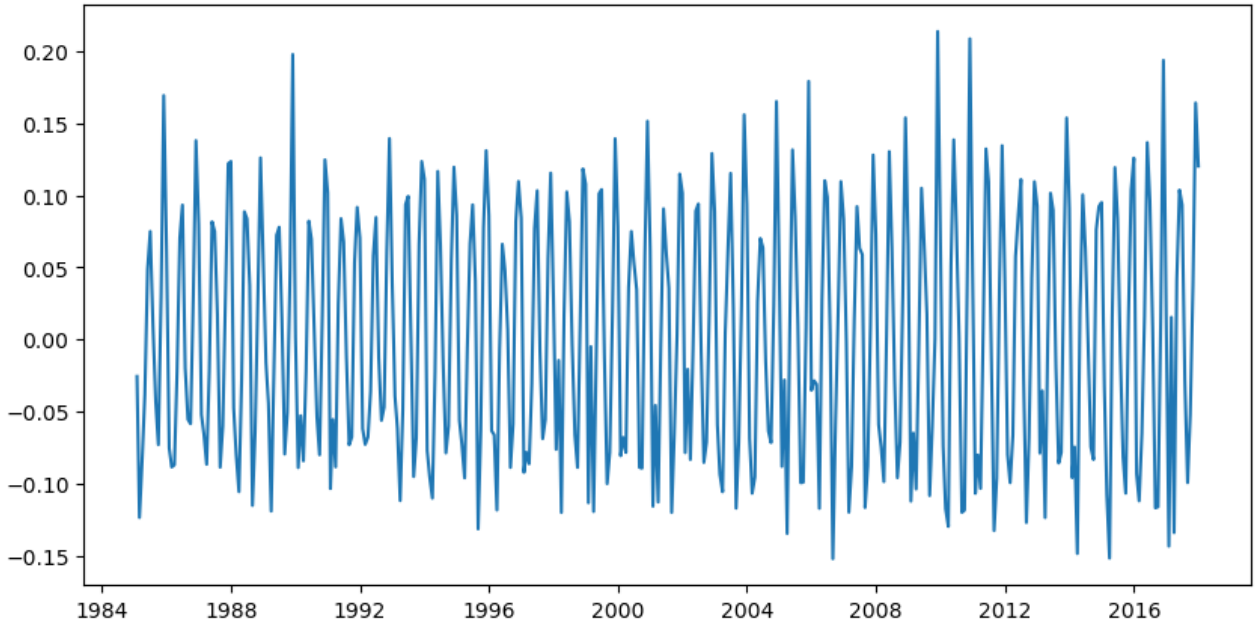
Out[31]:
```
[<matplotlib.lines.Line2D at 0x7ffb40528b80>]
```



Appendix 5

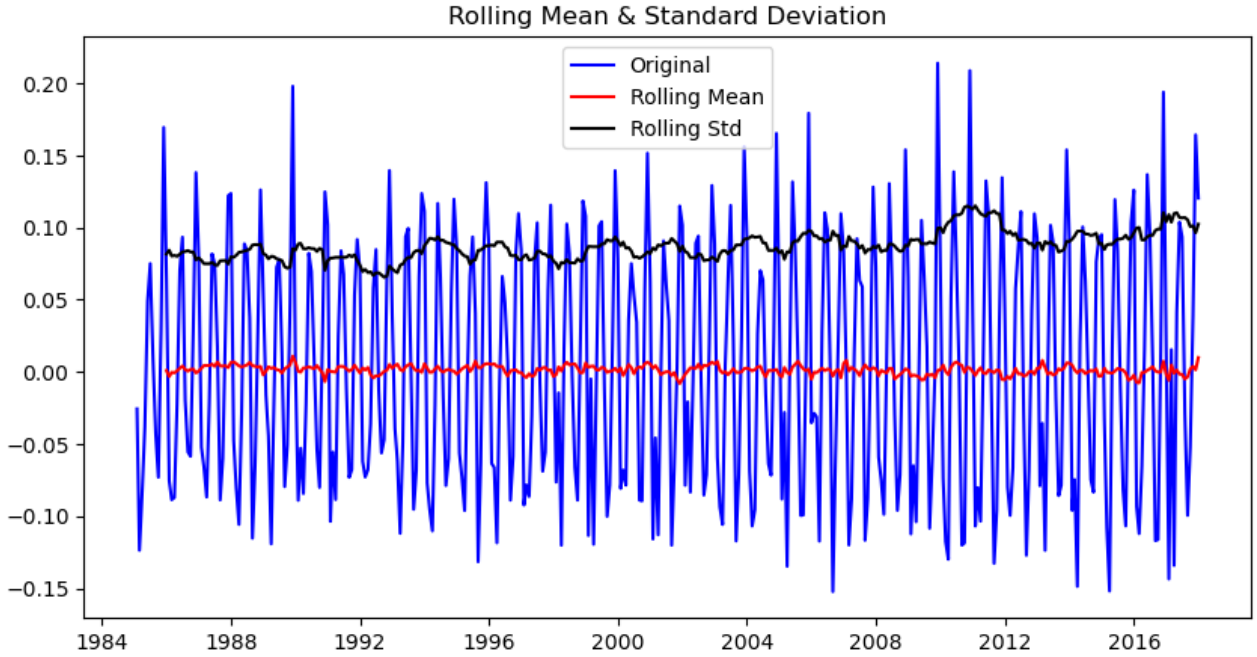
In [32]:
```
#find the first difference
first_difference = df_log.diff().dropna()
plt.figure(figsize=(10,5))
plt.plot(first_difference)
```

Out[32]:
```
[<matplotlib.lines.Line2D at 0x7ffb404664a0>]
```



Appendix 6

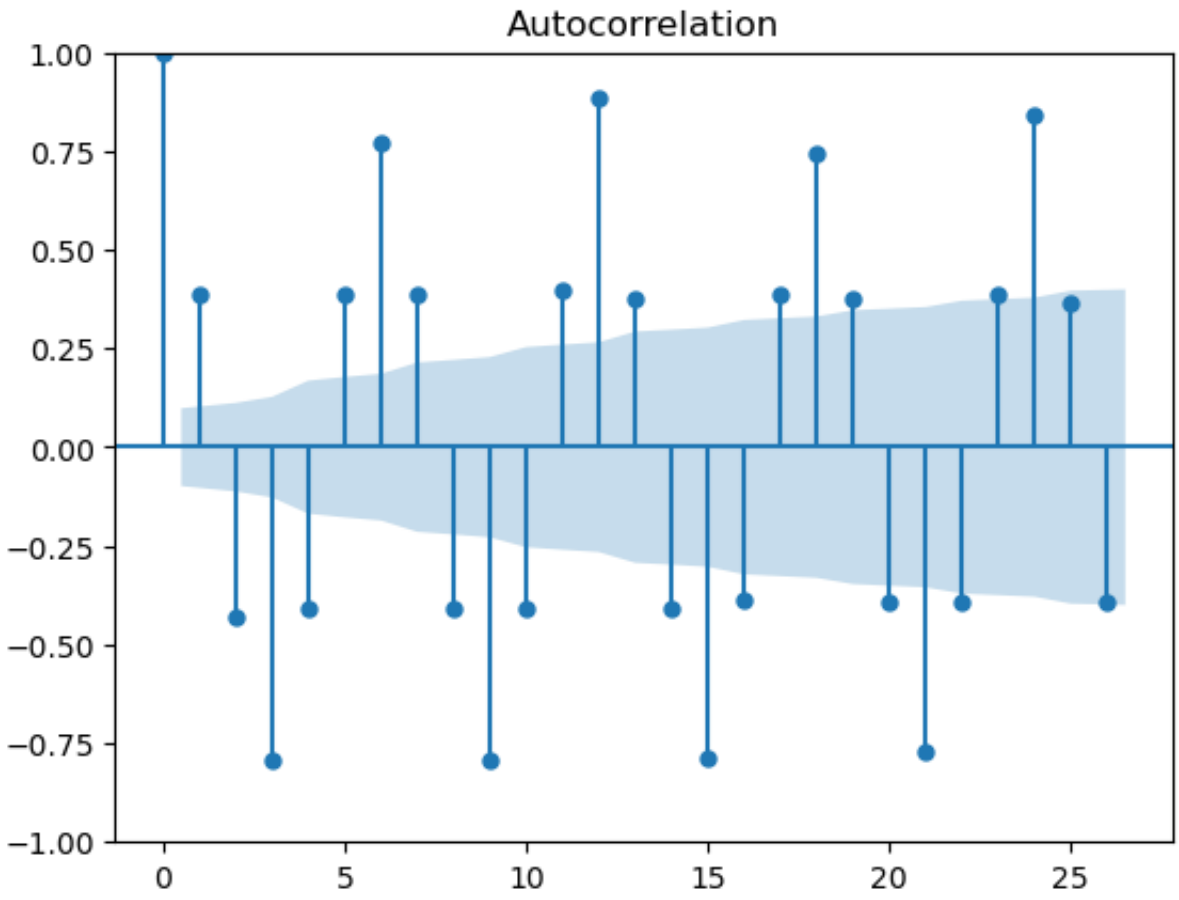
In [33]:
```
#test
analyze_stationarity(first_difference)
```

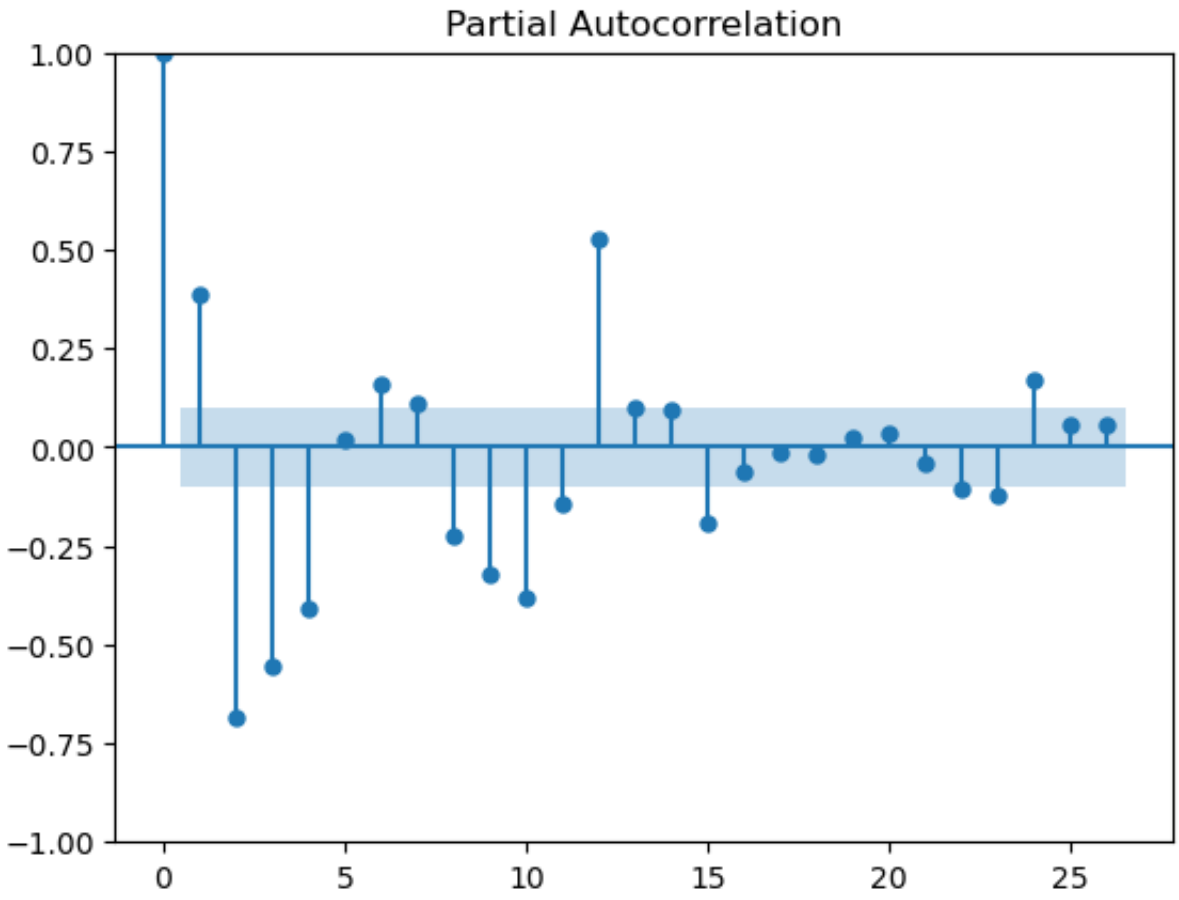



```
Results of Dickey-Fuller Test:
Test Statistic                -6.748333e+00
p-value                        2.995161e-09
#Lags Used                     1.400000e+01
Number of Observations Used    3.810000e+02
Critical Value (1%)           -3.447631e+00
Critical Value (5%)           -2.869156e+00
Critical Value (10%)          -2.570827e+00
dtype: float64
```

Appendix 7

In [34]:
```
# Plot Autocorrelation Function (ACF)
plot_acf(first_difference)
plt.show()

# Plot Partial Autocorrelation Function (PACF)
plot_pacf(first_difference)
plt.show()
```




Partial Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25

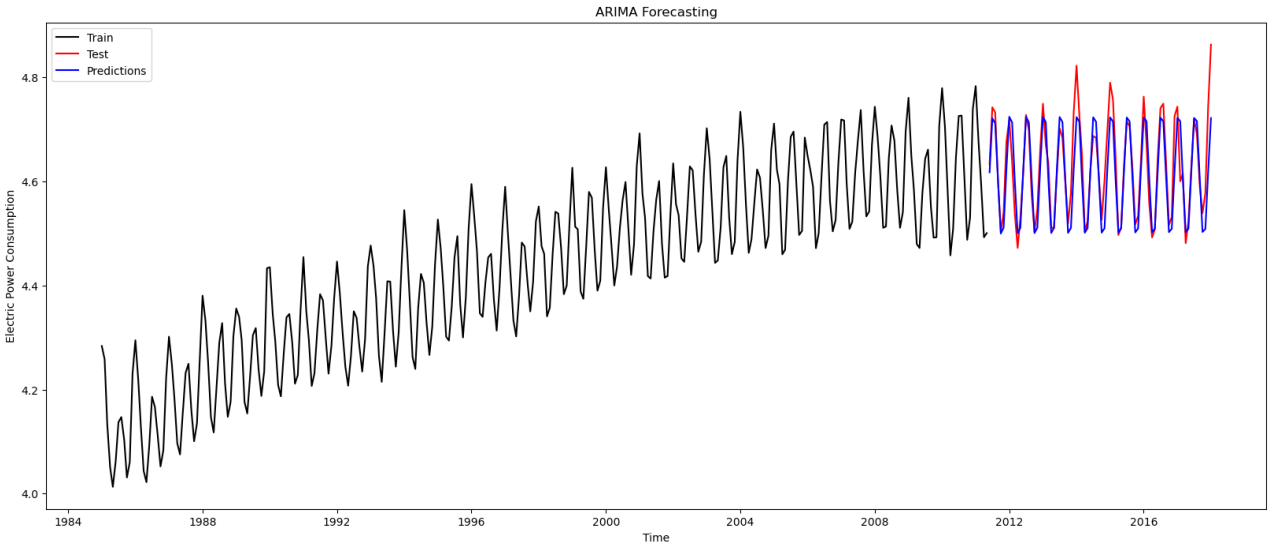
Appendix 8

In [17]:
```
# Define the p, d, and q ranges
p_range = range(0, 5) # for AR terms
d_range = range(0, 5) # for differencing
q_range = range(0, 5) # for MA terms

# Generate all different combinations of p, d and q
pdq = list(itertools.product(p_range, d_range, q_range))

# Hold the best values
best_aic = np.inf
best_pdq = None

# Grid search
for combo in pdq:
    try:
        # Create and fit ARIMA model
        model = ARIMA(df_log, order=combo)
        model_fit = model.fit()

        # Check if this AIC is the lowest found so far
        if model_fit.aic < best_aic:
            best_aic = model_fit.aic
            best_pdq = combo
    except:
        continue

print(f'Best ARIMA parameters are: {best_pdq} with AIC: {best_aic}')
```

```
Best ARIMA parameters are: (4, 1, 4) with AIC: -1511.5420262246466
```

Appendix 9

In [35]:
```
#split
train, test = train_test_split(df_log, test_size=0.2, shuffle=False)

#order
order = (4, 1, 4)

#train and test
y_train = train['Value']
y_test = test['Value']

#fit
final_model = ARIMA(y_train, order=order)
final_model_fit = final_model.fit()

#predit
predictions = final_model_fit.predict(start=test.index.min(), end=test.index

#plot
plt.figure(figsize=(20, 8))
plt.plot(train.index, train['Value'], color='black', label='Train')
plt.plot(test.index, test['Value'], color='red', label='Test')
plt.plot(test.index, predictions, color='blue', label='Predictions')
plt.ylabel('Electric Power Consumption')
plt.xlabel('Time')
plt.title('ARIMA Forecasting')
plt.legend()
plt.show()
```



In [36]:
```
print(final_model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
===
Dep. Variable:                   Value   No. Observations:
317
Model:                 ARIMA(4, 1, 4)   Log Likelihood                  613.
757
```

```
Date:                Thu, 15 Jun 2023   AIC                          -1209.
514
Time:                        04:46:06   BIC                          -1175.
713
Sample:                    01-01-1985   HQIC                         -1196.
011
                         - 05-01-2011
Covariance Type:                 opg
==============================================================================
===
                 coef    std err          z      P>|z|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
ar.L1          0.5459      0.174      3.146      0.002       0.206       0.
886
ar.L2         -0.0062      0.120     -0.051      0.959      -0.242       0.
230
ar.L3         -0.9938      0.120     -8.267      0.000      -1.229      -0.
758
ar.L4          0.5382      0.131      4.100      0.000       0.281       0.
795
ma.L1         -0.7678      0.178     -4.302      0.000      -1.118      -0.
418
ma.L2          0.0130      0.184      0.071      0.944      -0.347       0.
373
ma.L3          0.9693      0.170      5.715      0.000       0.637       1.
302
ma.L4         -0.7817      0.162     -4.835      0.000      -1.099      -0.
465
sigma2         0.0012      0.000      8.396      0.000       0.001       0.
001
==============================================================================
========
Ljung-Box (L1) (Q):                   0.04   Jarque-Bera (JB):
9.75
Prob(Q):                              0.84   Prob(JB):
0.01
Heteroskedasticity (H):               1.08   Skew:
0.14
Prob(H) (two-sided):                  0.70   Kurtosis:
3.81
==============================================================================
========

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (comp
lex-step).
```

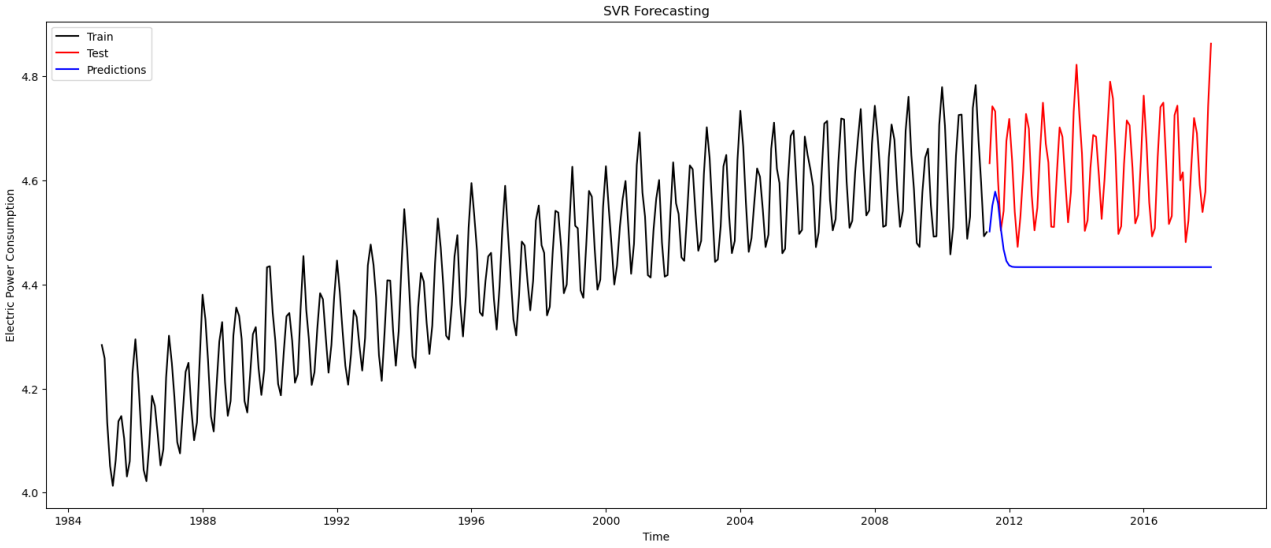
Appendix 10

In [37]:
```
#mse
mse = mean_squared_error(y_test, predictions)
rmse = sqrt(mse)
mae = mean_absolute_error(y_test, predictions)

print('Mean Absolute Error:', mae)
print('Mean Squared Error:', mse)
print('Root Mean Squared Error:', rmse)
```

```
Mean Absolute Error: 0.03173803985770368
Mean Squared Error: 0.0019869450872557507
Root Mean Squared Error: 0.04457516222354946
```

Appendix 11

In [38]:
```
y_train = train['Value']
y_test = test['Value']

# scale
scaler = StandardScaler()
train_scaled = scaler.fit_transform(np.array(y_train).reshape(-1, 1))
test_scaled = scaler.transform(np.array(y_test).reshape(-1, 1))

# reshape
X_train = np.arange(len(train_scaled)).reshape(-1, 1)
X_test = np.arange(len(train_scaled), len(train_scaled) + len(test_scaled)).

# fit model
model = SVR(kernel='rbf', C=1e3, gamma=0.1)
model.fit(X_train, train_scaled.ravel())

# predictions
predictions_scaled = model.predict(X_test)
predictions = scaler.inverse_transform(predictions_scaled.reshape(-1, 1))

#plot
plt.figure(figsize=(20, 8))
plt.plot(train.index, y_train, color='black', label='Train')
plt.plot(test.index, y_test, color='red', label='Test')
plt.plot(test.index, predictions, color='blue', label='Predictions')
plt.ylabel('Electric Power Consumption')
plt.xlabel('Time')
plt.title('SVR Forecasting')
plt.legend()
plt.show()
```



Apendix 12

In [39]:
```
#mse
mse = mean_squared_error(y_test, predictions)
rmse = sqrt(mse)
mae = mean_absolute_error(y_test, predictions)

print('Mean Absolute Error:', mae)
print('Mean Squared Error:', mse)
print('Root Mean Squared Error:', rmse)
```

```
Mean Absolute Error: 0.18720633468430337
Mean Squared Error: 0.04389069671047716
Root Mean Squared Error: 0.20950106613207764
```