Time Series Analysis of Kazakhstan Energy data using ARIMA model and methods from topology

July 13, 2024

By Alisher Mirmanov

Objective

The purpose of this project is my attempt at using persisten homology to time-series data. The data set I used is a time sereies represented by the renewable energy as a proportion to all energy sourses in the country found at stat.gov.kz. The analysis will try to model the behaviour of the time series by first doing a simple regression, asses it to get the benchmark error, then further complicate the analysis by implementing time series model, in partilcular ARIMA, and use the MSE and Akaikes test to test the model fit. Then we forecast the expected growth in the 5 years ahead. Lastly, the methods in Topological Data analysis are used to analyse the dataset. These methods look at the shape of the data, as a collection of simplicial complexes and asses the Betti number of the resultant metric space, that changes based on the open cover of the set(radious of open balls). While these methods are mostly applied in classification tasks, to, for instance, acquire additional clusters that conventional clustering algorithms fail to identify, there are interesting applications of TDA in time series analysis. In particular, one can transform the TS using time delay embedding operation, to trasnform the data set into d-dimensional space, on which later one can apply TDA methods to get the information about loops and voids of the data, that may indicate periodicity of the data, noise(for TS acquired from arious sensors), strcutural changes etc. In this case I only look at a simple TS data as a toy example, but similar methods can be applied to broader scale.

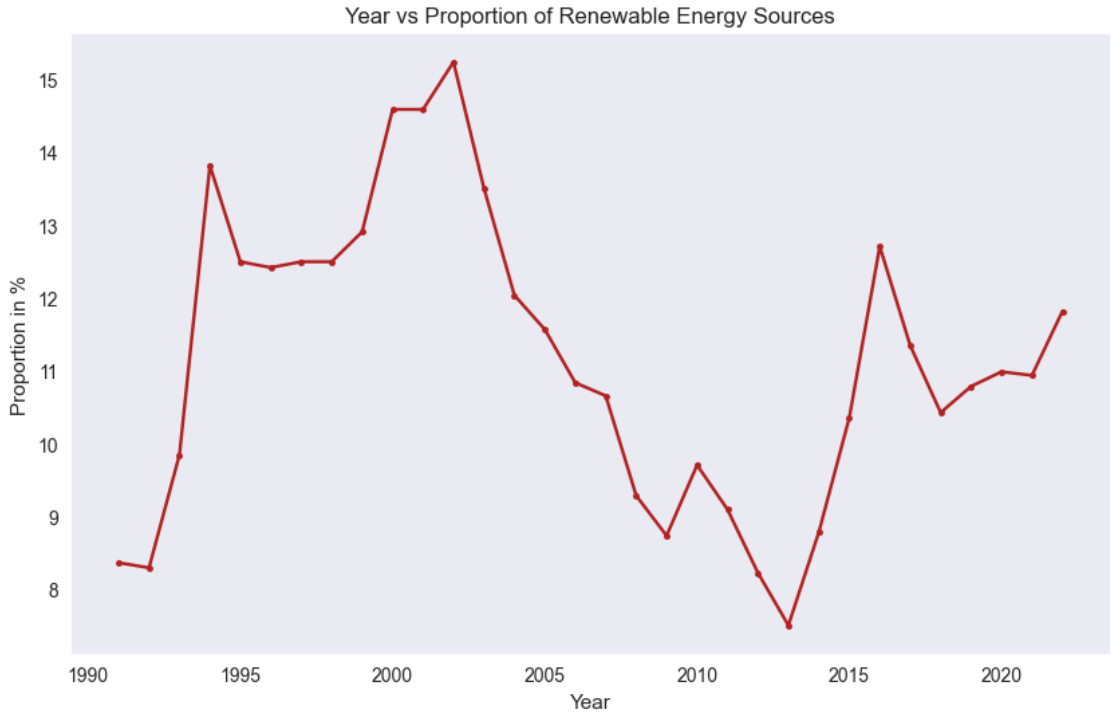
Step 1: Expected to growth using Regression and ARIMA model

[1]:
```
#Loading all the required Libraries
import numpy as np
import pandas as pd
import matplotlib.pylab as plt


from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn import linear_model


from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
plt.style.use('seaborn-v0_8')
```

```
import plotly.graph_objects as go
from gtda.time_series import SingleTakensEmbedding
from gtda.plotting import plot_point_cloud
from gtda.homology import VietorisRipsPersistence

import warnings
warnings.filterwarnings('ignore')
```

[4]:
```
#reading the dataset
re = pd.read_csv('RE.csv')
```

[5]:
```
#assign the variables to the appropriate values in df
years = re.iloc[5, 3:].values
proportion = re.iloc[6, 3:].values
proportion = pd.to_numeric(proportion, errors='coerce')
years = pd.to_numeric(years, errors='coerce')

# Plotting the graph
plt.figure(figsize=(10, 6))
plt.plot(years, proportion, marker = '.', color = 'Firebrick')
plt.title('Year vs Proportion of Renewable Energy Sources')
plt.xlabel('Year')
plt.ylabel('Proportion in %')
plt.grid(False)
plt.show()
```



Before fitting the time series model, let us asses how the linear regression performs to benchmark the error.

[6]:
```
#turing into 2d array for scikit lib and split intro train and test sets
years2D = years.reshape(-1,1)
X_train, X_test, y_train, y_test = train_test_split(years2D, proportion,␣
 ↪test_size=0.33)

regression = linear_model.LinearRegression()
regression.fit(X_train, y_train)
y_reg_line = regression.predict(X_test)
plt.scatter(years, proportion)
plt.scatter(X_test, y_test)
plt.plot(X_test, y_reg_line)
plt.grid(False)
plt.xticks(rotation=45)
plt.show()
```

[7]:
```
#getting the means square error on the test set
mse = mean_squared_error(y_test, y_reg_line)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 3.9248689905612006
```

[10]:
```
#This code Draws a prediction for the 20 years ahead

# future = np.array([2022+i for i in range(1, 20)])
# future = future.reshape(-1, 1)
# inference = regression.predict(future)
# plt.plot(X_test, y_reg_line,)
# plt.plot(years, proportion)
# plt.scatter(future, inference)
# plt.grid(False)
```

It is clear, that the linear regression is an inappropriate model for describing the data recieved, due to hihg non-linearity of the data. One option to model such behaviour could be Generative Additive Models, like local regression, that are likely to model such data well, but are rather unsuited for extrapolation beyond the observed time range(not suited for forecasting process). One shall choose a model, appropriate for such extrapolation, such as various time series models.

1. First we using the Augmented DF test we check if the time series is stationary.
2. We need to plot the ACF and PACF functions, which will give us some idea about the order of AR and MA components of the models.
3. We then make the series statinoary by differencing and again check using ADF test.
4. Then we fit the ARIMA(p,d,q) on the trainting set to find the parameters of the model(phis and thetas) and test it on the test set.

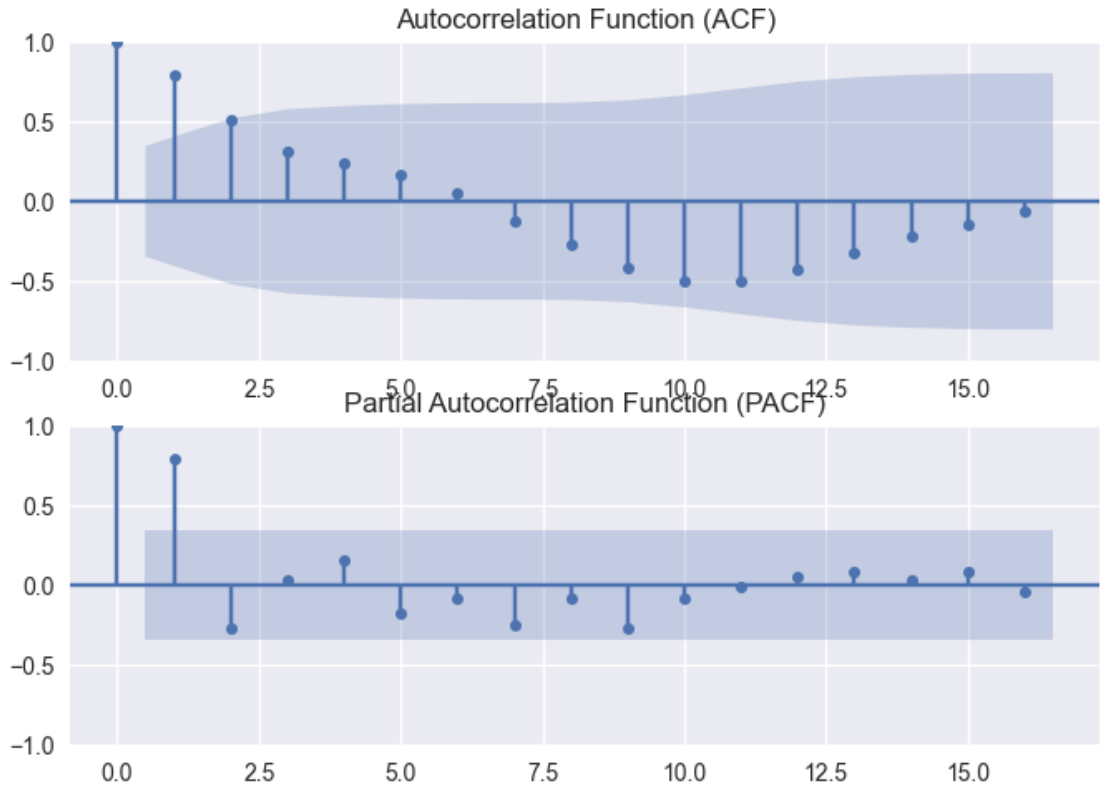
[84]:
```
check_stat = adfuller(proportion)
# Extract test statistics
adf_statistic = check_stat[0]
p_value = check_stat[1]

# Print results
print(f"ADF Statistic: {adf_statistic}")
print(f"P-value: {p_value}")

# Check for stationarity
if p_value < 0.05:
    print("The series is stationary.")
else:
    print("The series is not stationary.")
```

```
ADF Statistic: -2.469788755370836
P-value: 0.12299913229972337
The series is not stationary.
```

The results suggest that the time series is non stationary, which is expected. ARIMA model is likely to be a suitable model for the task. Start by aquiring ACF and PACF diagrams and constructing

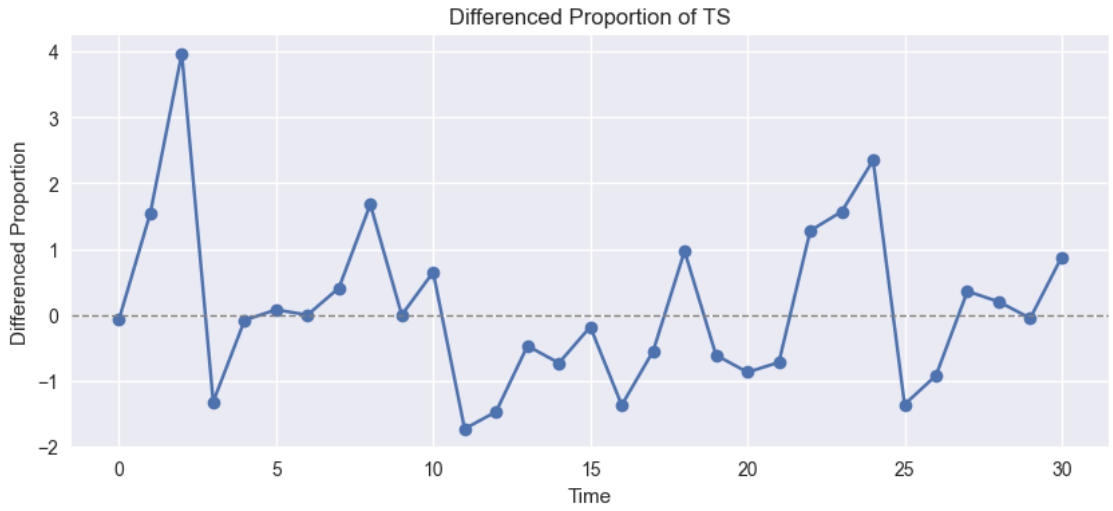
second order difference graph to make the series stationary.

[86]:
```
#plotting acf and pacf to anaylze he time series data
fig, ax = plt.subplots(2, 1)

plot_acf(proportion, ax=ax[0])
ax[0].set_title('Autocorrelation Function (ACF)')
plot_pacf(proportion, ax=ax[1])
ax[1].set_title('Partial Autocorrelation Function (PACF)')

plt.show()
```



[87]:
```
#Checking if the first differencing makes the time series stationary
proportion_diff = np.diff(proportion)

# Plot differenced time series
plt.figure(figsize=(10, 4))
plot = plt.plot(proportion_diff, marker='o')
plt.title('Differenced Proportion of TS')

plt.xlabel('Time')
plt.ylabel('Differenced Proportion')
```

```
plt.axhline(y=0, color='gray', linestyle='--', linewidth=1)

plt.show()
```



[88]:
```
check_stat_1 = adfuller(proportion_diff)
# Extract test statistics
adf_statistic_1 = check_stat_1[0]
p_value_1 = check_stat_1[1]

# Print results
print(f"ADF Statistic: {adf_statistic_1}")
print(f"P-value: {p_value_1}")

# Check for stationarity
if p_value_1 < 0.05:
    print("The series is stationary.")
else:
    print("The series is not stationary.")
```

```
ADF Statistic: -4.472821198030668
P-value: 0.00020211279632142786
The series is stationary.
```

The results, in particular the p-value of 0.002, suggest that the first differencing is sufficient to produce a stationary time series(0.002 «0.05), thus we try model the time series using different orders and see which has smaller MSE and AIC on the test set.

[89]:
```
train_years = years[:25]
train_proportions = proportion[:25]
test_years = years[25:]
```

```
test_proportions = proportion[25:]
```

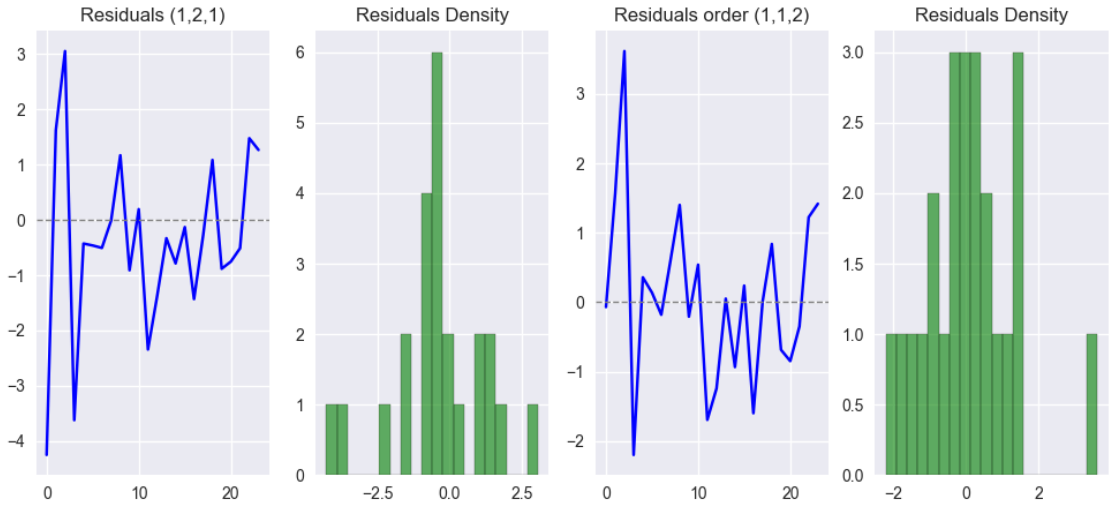
[118]:
```
#Building ARIMA on train data
for p in range(1, 3):
    for q in range(1, 3):
        for d in range (1, 3):
            model = ARIMA(train_proportions, order = (p, d, q))
            model_fit = model.fit()
            aic = model_fit.aic
            forecast = model_fit.forecast(steps=7)
            mse = mean_squared_error(test_proportions, forecast)
            print( p, d, q, "order: mse:", mse, "AIC : ", aic)
```

```
1 1 1 order: mse: 1.0908484325656347 AIC :  84.34839065840445
1 2 1 order: mse: 0.6865582808054934 AIC :  84.93481433987407
1 1 2 order: mse: 0.689961784576985 AIC :  86.15674551225445
1 2 2 order: mse: 0.8270016651286544 AIC :  86.82469103622839
2 1 1 order: mse: 1.3961708531011963 AIC :  86.2235783504835
2 2 1 order: mse: 0.961537713069326 AIC :  86.71180590589039
2 1 2 order: mse: 2.425227294658154 AIC :  86.40631392501142
2 2 2 order: mse: 15.76979645438915 AIC :  87.0444690309007
```

The results clearly already suggest that the ARIMA model is way more accurate that linear regression, and additionally, we see that the order (1,2,1) and (1,1,2) produce the best results with (1,2,1) performing slightly better on AIC test. By constructing the residual density diagram(see below) we see that the (1,2,1) order also better captures the model as it closely resembles the Gaussian density, making it a better fit for the analysis.

[119]:
```
model_1= ARIMA(train_proportions, order = (1, 2, 1))
model_2 = ARIMA(train_proportions, order = (1, 1, 2))
model_fit_1= model_1.fit()
model_fit_2= model_2.fit()
residuals = model_fit_1.resid[1:]  # Skip the first element if needed
residuals_2 = model_fit_2.resid[1:]


# Create a figure with 1 row and 2 columns of subplots
fig, ax = plt.subplots(1, 4, figsize=(12, 5))

# Plot the residuals as a time series
ax[0].set_title('Residuals (1,2,1)')
ax[0].plot(residuals, label='Residuals', color='blue')
ax[0].axhline(y=0, color='gray', linestyle='--', linewidth=1)  # Add a␣
 ↪horizontal line at y=0


ax[2].set_title('Residuals order (1,1,2)')
ax[2].plot(residuals_2, label='Residuals', color='blue')
```

```
ax[2].axhline(y=0, color='gray', linestyle='--', linewidth=1)  # Add a␣
↪horizontal line at y=0




# Histogram
ax[1].hist(residuals, bins=20, color='green', alpha=0.6, edgecolor='black')
ax[1].set_title('Residuals Density')

# Histogram
ax[3].hist(residuals_2, bins=20, color='green', alpha=0.6, edgecolor='black')
ax[3].set_title('Residuals Density')

# Show the plot
plt.show()
```



All the results suggest that the ARIMA(1,2,1) performs best on the test set, and as such we use it as the model for forecasting the future values

**

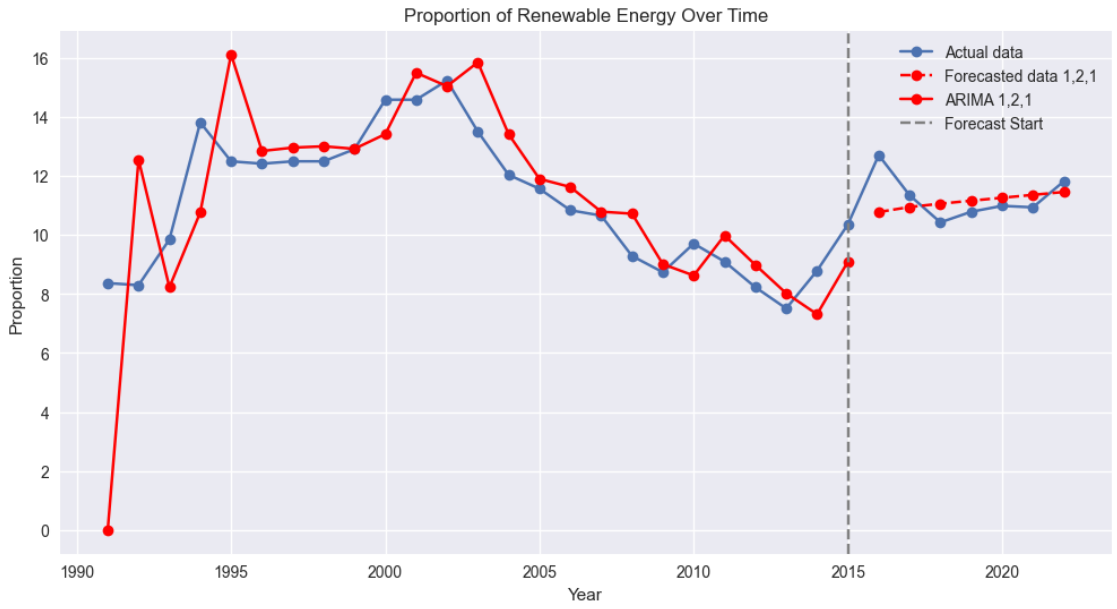
Training and Testing the data Graph

**

[120]:
```
test_forecast_1 = model_fit_1.forecast(steps=7)
#test_forecast_2 = model_fit_2.forecast(steps=7)
```

[121]:
```
plt.figure(figsize=(12, 6))
fitted_values_1t = model_fit_1.fittedvalues
```

```
#fitted_values_2t = model_fit_2.fittedvalues

plt.plot(years, proportion, label='Actual data', marker='o')
plt.plot(test_years, test_forecast_1, label='Forecasted data 1,2,1', marker='o',␣
↪linestyle='--', color = 'red')
#plt.plot(test_years, test_forecast_2, label='Forecasted data for 1,1,2',␣
↪marker='x', linestyle='--', color = 'green')
plt.plot(years[:25], fitted_values_1t, label='ARIMA 1,2,1', marker='o', color =␣
↪'red')
#plt.plot(years[:25], fitted_values_2t, label='ARIMA 1,1,2', marker='o',color =␣
↪'green')
plt.axvline(x=train_years[-1], color='gray', linestyle='--', label='Forecast␣
↪Start')
plt.xlabel('Year')
plt.ylabel('Proportion')
plt.title('Proportion of Renewable Energy Over Time')
plt.legend()
plt.show()
```



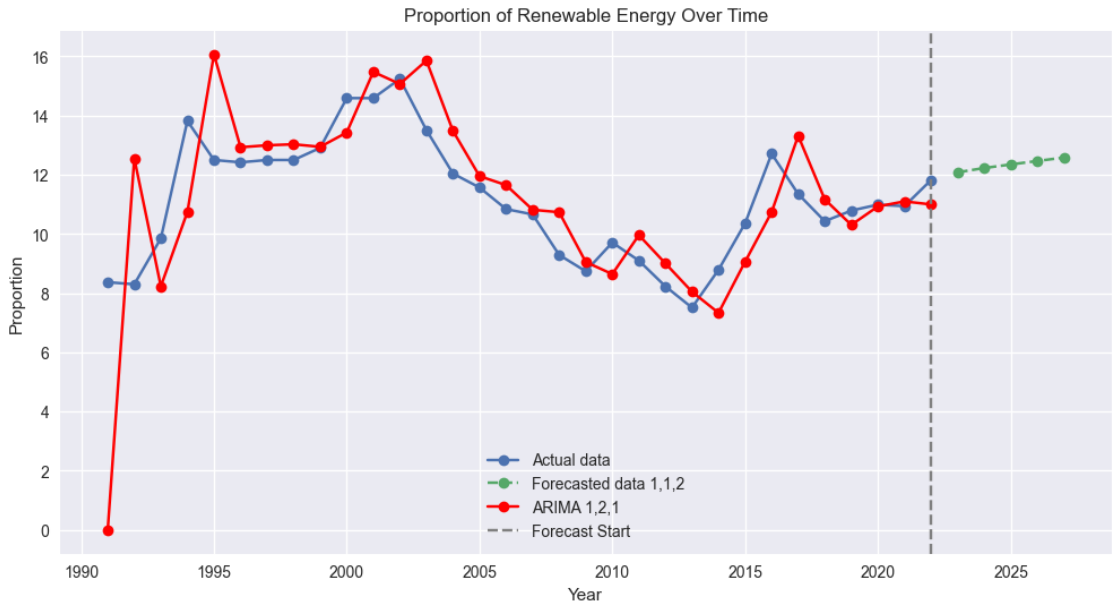
**

Forecasting 5 years ahead

**

[9]:
```
#training the model on the whole set of values and forecasting 10 yrs into the␣
↪future
```

```
model_forecast_1 = ARIMA(proportion, order = (1, 2, 1))
#model_forecast_2 = ARIMA(proportion, order = (1, 1, 2))
model_fit_forecast_1 = model_forecast_1.fit()
#model_fit_forecast_2 = model_forecast_2.fit()


fitted_values_1 = model_fit_forecast_1.fittedvalues
#fitted_values_2 = model_fit_forecast_2.fittedvalues



forecast_1 = model_fit_forecast_1.forecast(steps=5)
#forecast_2 = model_fit_forecast_2.forecast(steps=5)


plt.figure(figsize=(12, 6))

plt.plot(years, proportion, label='Actual data', marker='o')
plt.plot([2021+i for i in range(2,7)],forecast_1, label='Forecasted data 1,1,2',␣
 ↪marker='o', linestyle='--')
#plt.plot([2021+i for i in range(2,7)], forecast_2, label='Forecasted data for␣
 ↪1,2,1', marker='x', linestyle='--')
plt.plot(years, fitted_values_1, label='ARIMA 1,2,1', marker='o', color = 'red')
#plt.plot(years, fitted_values_2, label='ARIMA 1,1,2', marker='o',color =␣
 ↪'green')
plt.axvline(x=2022, color='gray', linestyle='--', label='Forecast Start')


plt.xlabel('Year')
plt.ylabel('Proportion')
plt.title('Proportion of Renewable Energy Over Time')
plt.legend()
plt.show()

print("In the year 2032, the expected proportion of renewable energy sources to␣
 ↪total energy sourses is expected to be", forecast_1[-1], "%")
```




```
In the year 2032, the expected proportion of renewable energy sources to total
energy sourses is expected to be 12.581262347703372 %
```

**

Topology

**

**

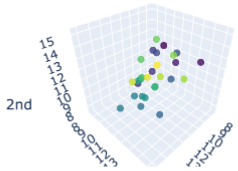
Time Delay Embedding of the Time Series

**

Now we use topological methods to analyze the time series. We start by using the time embedding described in detail here: https://zilliz.com/learn/time-series-embedding-data-analysis. As an illustrative example we start with time delay of 2, and then analyze a range of delays later.

[16]:
```
embedding_dimension = 3
embedding_time_delay = 2
stride = 1

embedder = SingleTakensEmbedding(
    parameters_type="fixed",
    n_jobs=2,
    time_delay=embedding_time_delay,
    dimension=embedding_dimension,
    stride=stride,
)
```

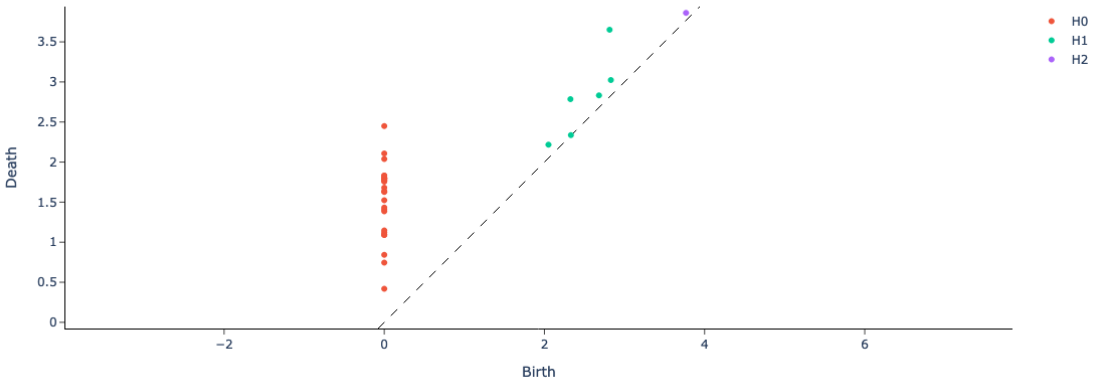
```
y_embedded = embedder.fit_transform(proportion)
plot_point_cloud(y_embedded)
```



[17]:
```
y_embedded = y_embedded[None, :, :]
```

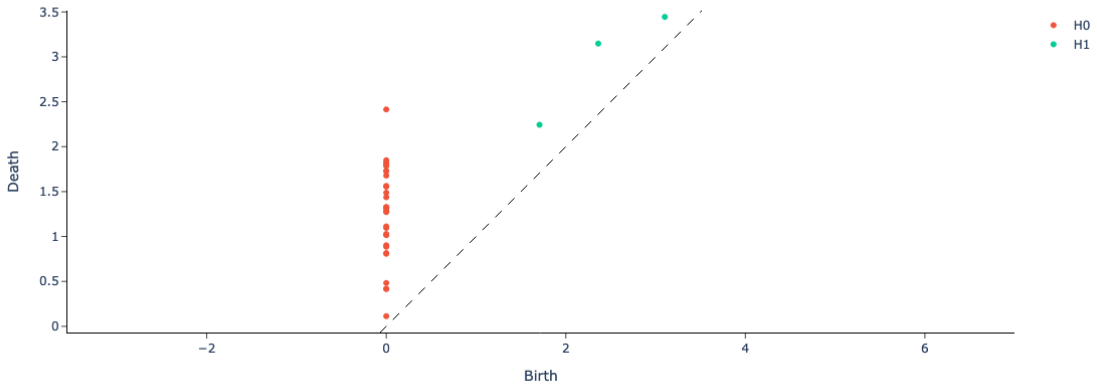
[20]:
```
# 0 - connected components, 1 - loops, 2 - voids
homology_dimensions = [0, 1, 2]

nonperiodic_persistence = VietorisRipsPersistence(
    homology_dimensions=homology_dimensions, n_jobs=6
)
print("Persistence diagram for nonperiodic signal")
nonperiodic_persistence.fit_transform_plot(y_embedded);
```

```
Persistence diagram for nonperiodic signal
```

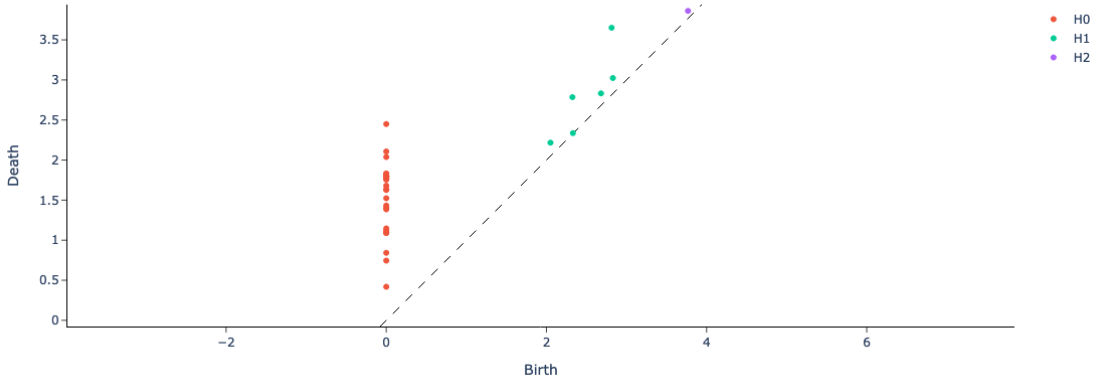


[21]:
```
#Try different time embeddings and look at their persistance diagram


embedding_dimension = 3
embedding_time_delay = [i for i in range(1,7)]
stride = 1

for delay in embedding_time_delay:
    embedder = SingleTakensEmbedding(
        parameters_type="fixed",
        n_jobs=2,
        time_delay=delay,
        dimension=embedding_dimension,
        stride=stride,
    )

    y_embedded = embedder.fit_transform(proportion)
    plot_point_cloud(y_embedded)

    y_embedded = y_embedded[None, :, :]


    homology_dimensions = [0, 1, 2]


    print("Persistence diagram for delay", delay)
    nonperiodic_persistence.fit_transform_plot(y_embedded);
    persistence = VietorisRipsPersistence(
        homology_dimensions=homology_dimensions, n_jobs=6
    )
```

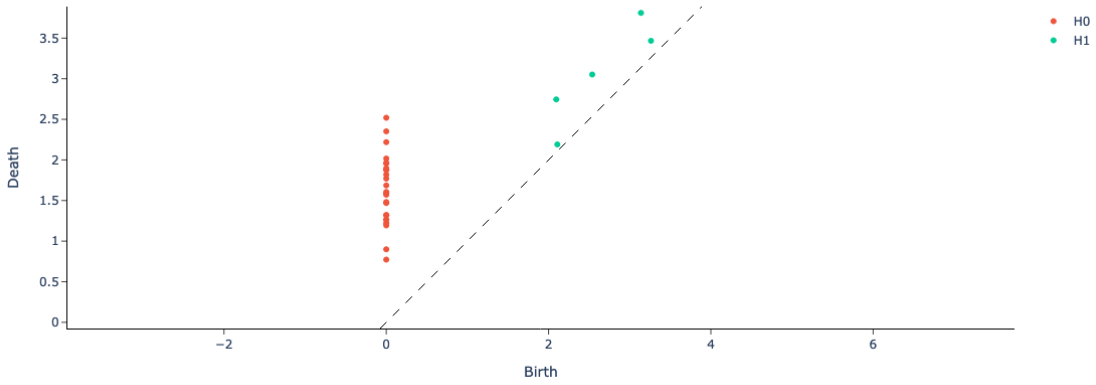
```
Persistence diagram for delay 1
```



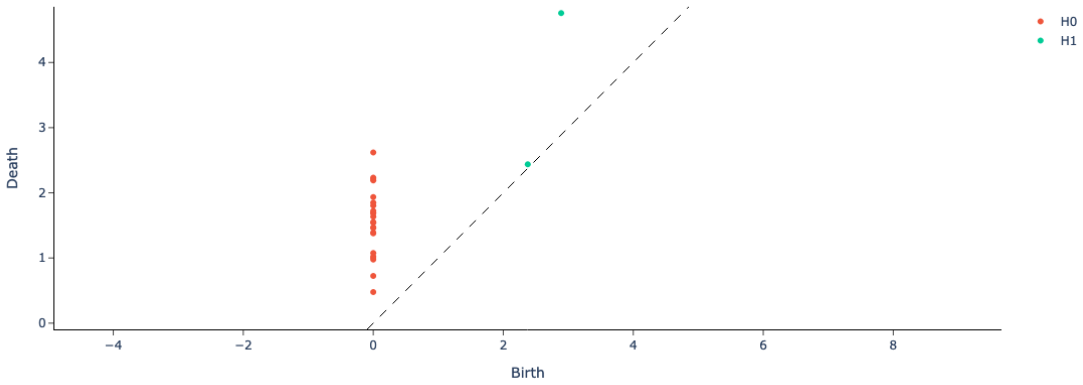
Persistence diagram for delay 2



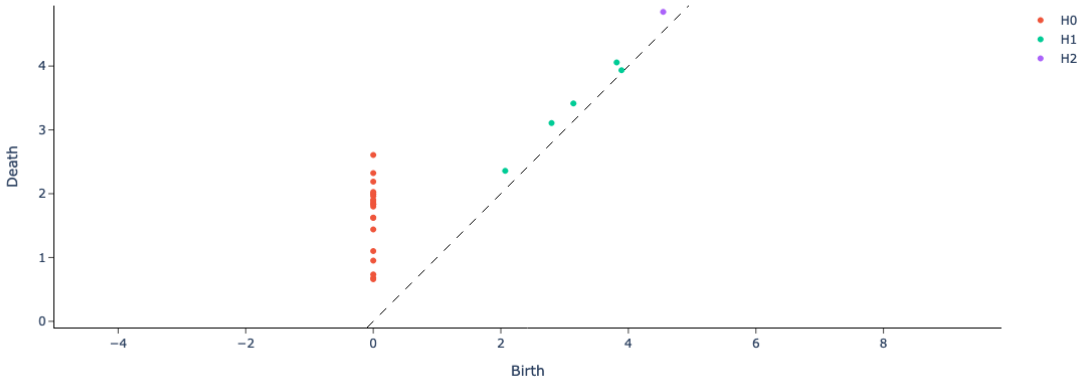
Persistence diagram for delay 3



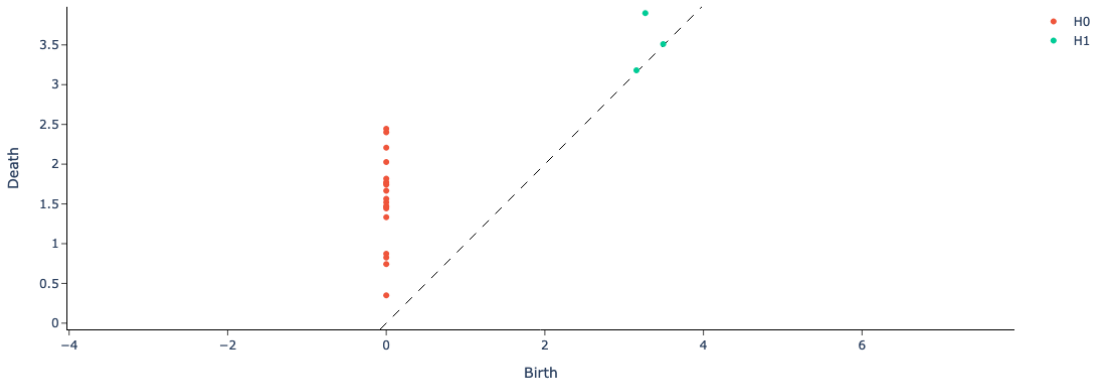
Persistence diagram for delay 4



Persistence diagram for delay 5



Persistence diagram for delay 6



Analysing the results and conclusion

The homology diagram can give us hints about the structure of the time series data. In particular, the presence of long-lived 0-dimensional features (H_0 (connected components)) across the whole range of time delays might indicate that, for a variety of scales (radii of open balls around points), the connected components stay connected. This suggests that the time series data is not highly erratic or fragmented and follows a coherent path with recurring patterns.

The presence of at least 2-3 H_1 or single-dimensional homology (loops) across the range of delays might strongly indicate some seasonality in the data, which the ARIMA model didn't capture. As such, the forecast should be fluctuating. Given these insights, it is worth considering augmenting the ARIMA model with additional components that capture cyclic patterns, such as seasonal decomposition or integrating features that reflect the persistent H_1 features. Lastly, we also notice the occurrence of occasional 2-dimensional homologies, which are voids, but these are very short-lived and occur over a very small span of radii. They might potentially reflect some intricate dynamics but likely are not significant for the analysis, so they shouldn't be considered indicators of any deeper structure. Overall, the analysis indicates that the data exhibits short-term fluctuations, cyclic nature, and long-term stability.

As such, TDA proves useful beyond its usual applications, such as clustering tasks. It allows us to capture seasonality and trends that are hard to detect using traditional methods, especially for a small data set like the one used in this investigation. A similar approach can be applied in various fields, including finance and scientific disciplines, to provide a unique perspective on analyzing complex data, including time series data, offering valuable insights that complement traditional analytical methods.