Predicting the population of Finland in 2025 with regression models

1 Introduction

Population data is essential for planning purposes. Why is information about the number of people living in a country, a region, or a town needed? Governments, for example, need to know how many inhabitants currently live in the country and how many there will be in the future. From this, they can plan and make better decisions regarding the construction of schools, hospitals, roads, etc. The goal of this project is to help the government to predict the population for future purposes.

The remainder of this paper is divided into 5 sections. Section 2 illustrates the problem formulation and discusses the dataset, selection of features, and label. Section 3 discusses the regression models, selection of the features and labels, and sets utilized in this project. The results obtained from implementing each model are illustrated in Section 4. Section 5 concludes the paper. Finally, Section 6 lists the materials used during the completion of this project and Section 7 is a implemented code.

2 Problem Formulation

The purpose of this paper is to implement machine learning methods to predict the population of Finland in 2025 under supervised machine learning. It is considered a supervised ML problem as the dataset contains labeled data points.

2.1 Dataset

There is a lot of data available online about the population of Finland. The sample dataset utilized in this project is the StatFin dataset published and updated every year by Statistics Finland. The StatFin dataset contains information on population, the number of immigrants and emigrants, deaths, live births, and other measurements for each year beginning from 1749. Each column is a data point in the dataset. There are 273 entries and only numeric data. The dataset is converted to .csv format in order to access it in Python3. Moreover, the code implements the Pandas library to get rid of the rows with missing values. Therefore, only 77 entries are utilized for the implementation of the regression models. In addition, all the data points are numerical, meaning they are integers. The table below describes all data points available in the dataset.

Variable	*Description*
Year	Numerical, Integer
Live Births	Numerical, Integer
Deaths	Numerical, Integer
Natural Increase	Numerical, Integer
Intermunicipal migration	Numerical, Integer
Immigration to Finland	Numerical, Integer
Emigration from Finland	Numerical, Integer
Net migration	Numerical, Integer
Marriages	Numerical, Integer

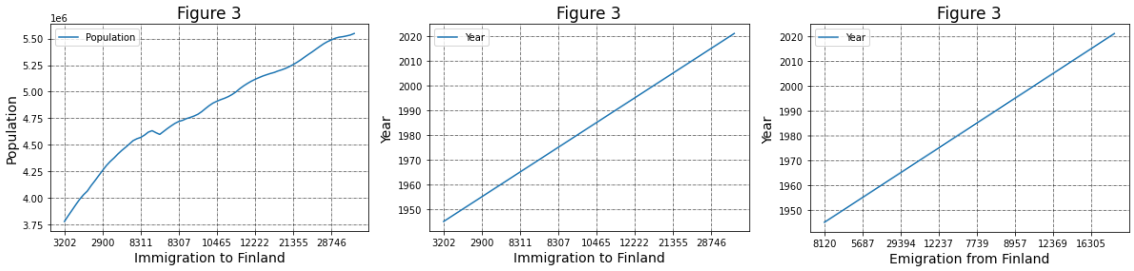
Divorces	Numerical, Integer
Total change	Numerical, Integer
Population	Numerical, Integer

Table1. Dataset content and its description.

3 Methods

3.1 Feature and label selection

The dataset contains 12 columns, however, not all of them contain the values for each entry. Therefore, after sorting the data, columns 'Live births', 'Deaths', 'Immigration to Finland', and 'Emigration from Finland' were selected as features. As can be seen from the figures below, there are relations between the population of Finland and some of the selected features. The column 'Population' was selected as a label. Moreover, it was decided that data points 'Year', 'Natural increase', 'Intermunicipal migration', 'Net migration', 'Marriages', 'Divorces', and 'Total change' will not be utilized. The reason for such a decision is that data points 'Natural increase', 'Net migration', and 'Total change' can be computed from already selected features, and data points 'Intermunicipal migration', 'Marriages', 'Divorces' are irrelevant in the scope of this problem.



3.2 Training, Validation, and Test sets

The dataset is split into training, validation, and test sets with a ratio of 60:20:20. This is achieved with the help of the "train_test_split" class. The reason for choosing such a ratio is the fact that it is utilized in the majority of Machine Learning problems. The training set contains 46 data points, whereas the validation set has 16 data points and the test set has 15 data points. Moreover, data points are distributed randomly into each set, therefore, there is no linear relationship between data points. It was achieved in order to avoid this relationship since this ML problem utilizes the statistics of the population of Finland. In addition, the test set contains data points that have not been used by either training or validation sets. Therefore, the outcome of the testing process is independent of the previous steps.

3.3 Linear Regression model

Linear regression is the first model that was chosen to implement the prediction algorithm. This model was chosen because it is generally a good starting point for describing continuous quantities such as population and there appears to be a linear relationship between the label and the features.

The hypothesis space for the linear regression model takes the form:

$$\mathcal{H}^{(n)} := \{h^{(\mathbf{w})} : \mathbb{R}^n \to \mathbb{R} : h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} \text{ with some parameter vector } \mathbf{w} \in \mathbb{R}^n\}.$$

The dataset contains data points characterized by feature vectors X and numeric label y. Therefore, the linear regression model learns a hypothesis out of this linear hypothesis space (Jung A., 2022).

Mean squared error is utilized as a loss function for this model. The reason for choosing the mean squared error is the fact that it is a widely used choice for the loss function for regression problems. In addition, it depends only on the prediction property of the features.

3.4 Polynomial Regression model

Polynomial Regression is the second model that was chosen to implement the prediction algorithm. This model was chosen due to the fact that many ML problems that involve data points that are characterized by numeric features and numeric labels use this model.

The hypothesis space for this model is constituted by polynomial maps (Jung A., 2022):

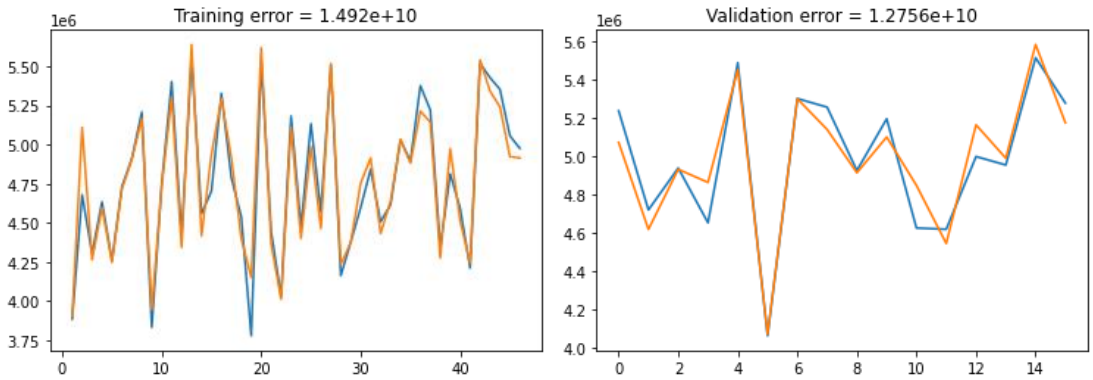
$$\mathcal{H}^{(n)}_{\text{poly}} = \{h^{(\mathbf{w})} : \mathbb{R} \to \mathbb{R} : h^{(\mathbf{w})}(x) = \sum_{j=1}^{n} w_j x^{j-1},$$
$$\text{with some } \mathbf{w} = (w_1, \ldots, w_n)^T \in \mathbb{R}^n\}.$$

Polynomial regression is a type of linear regression, therefore, the loss function for this model is mean squared error too. Polynomial regression learns a hypothesis h by minimizing the average squared error loss (Jung A., 2022).

4 Results

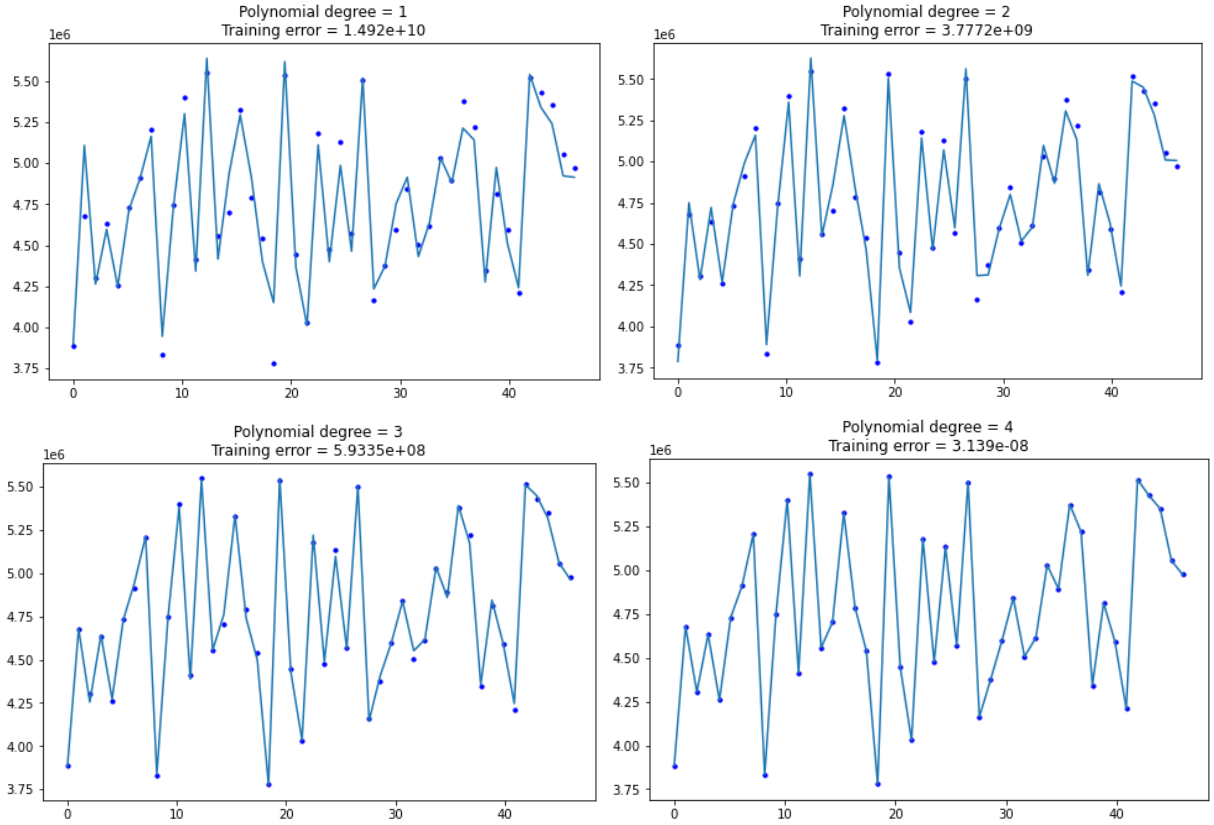
4.1 Linear Regression model

After implementing the linear regression model, the training and validation errors are computed. As seen from the figures below, the values of the errors are similar in some ways. However, the difference between the intercepts computed from both training and validation sets is very high. The intercept calculated from the training set is 5805315.104866516, whereas the validation set intercept is 2773723.2086173566. The big difference can be explained by the fact that data points in each set are selected randomly from the StatFin dataset. However, the validation intercept is very low regardless of it.

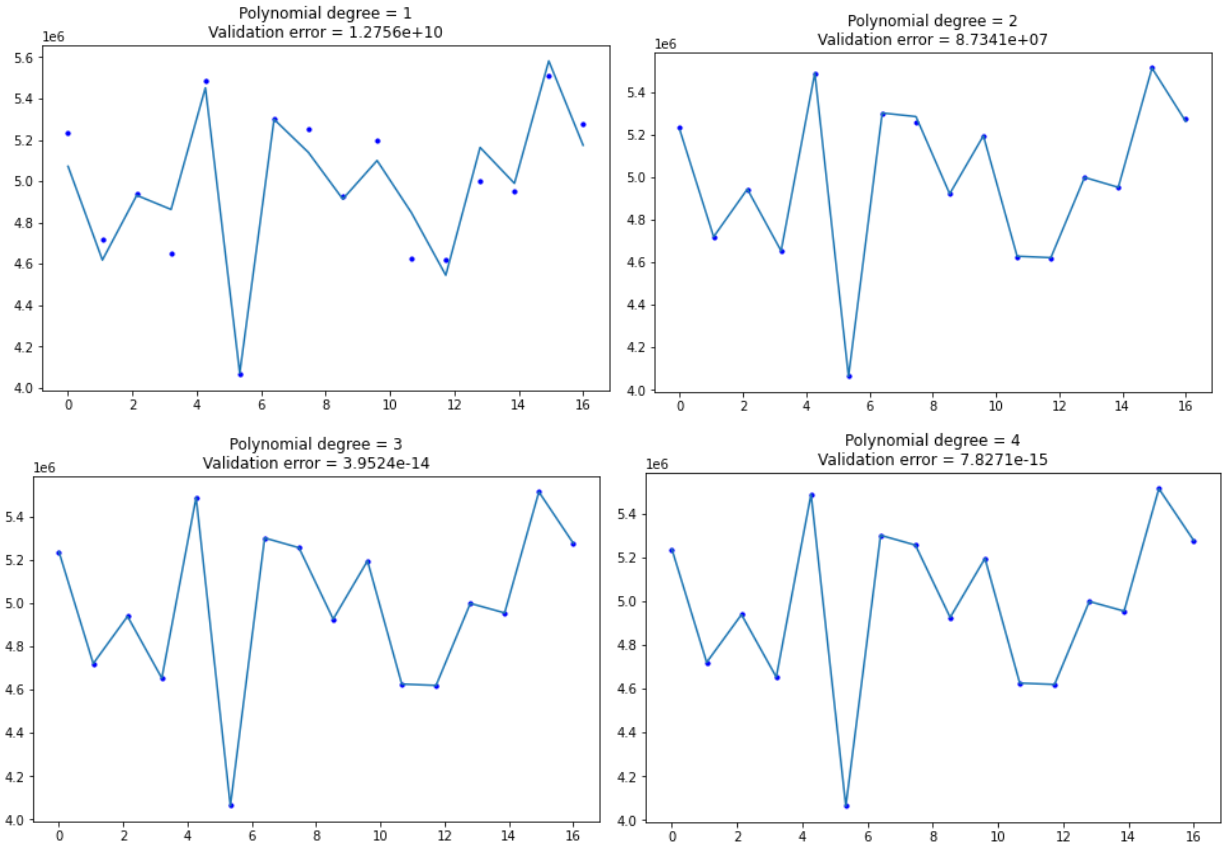


4.2 Polynomial Regression model

The results obtained from implementing this model are very scattered. Even though the polynomial regression model with a degree of 4 does a great job of predicting the population, models with degrees 1, 2, and 3 struggle with predicting the population value. Moreover, the training errors for the polynomial regression models with degrees 1, 2, and 3 are very high, whereas the training error for the model with degree 4 is very low.



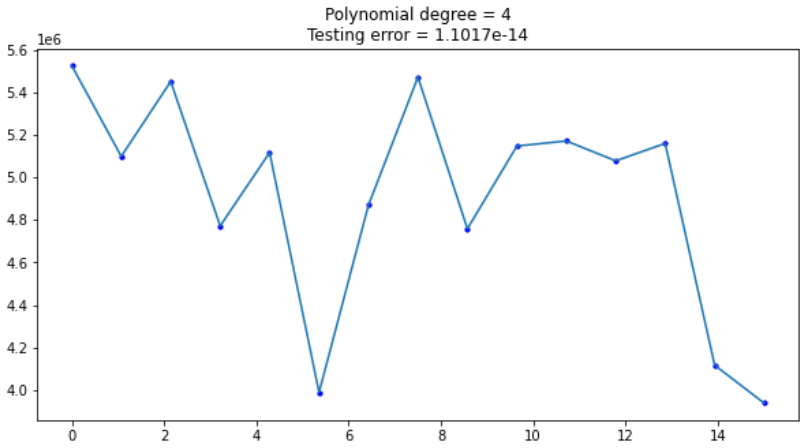
Even though the polynomial regression model with degree 4 is better than the other ones, all 4 models have to go through the validation process. The results obtained from this stage are unanticipated. From the figures below it can be seen that model with degree 3 has a considerably lower error compared to the same model tested with the training set. However, the polynomial regression model with degree 4 still performs better than the other models as it has a lower validation error.



4.3 Comparison, selection and testing

After implementing the models and computing the validation errors of each of them, the polynomial regression model with degree 4 is selected as the final model. This model performed better with both training and validation sets. In addition, it has a lower validation error than the training error. Even though it is a great result to collect, it can be explained by the sizes of the sets. The validation set has only 16 entries, whereas the training set has 46 entries. However, it does not affect the overall result obtained from testing all models.

After running the selected model with the test set, the test error is low meaning the content of the dataset does not affect the performance of the model. The test error is 1.1017e-14, therefore the model predicts the result correctly. In addition to the test error, the r^2 score of this model is calculated in order to test the overfitting of the model. The r^2 score is 1, which means the model fits perfectly.



5 Conclusion

To conclude, the best model to predict the population of Finland is the polynomial regression model with degree 3. This model performed extraordinarily throughout all stages of the project and succeeded to predict the desired result as there is no major difference in obtained error values. Moreover, selected features appear to be satisfying predictors.

Even though the model performs well, the reason for that could be the size of the dataset. As it was mentioned, the sizes of the sets are 46, 16 and 15. This can be interpreted as a very small dataset. However, considering the available data online, this is the maximum output that can be achieved from it.

There are improvements to make regarding the output of the project. The problem is predicting the population of Finland in 2025. However, it was not achieved in the scope of this paper. The reason for this is the lack of experience in implementing algorithms that forecast future values based on available data. Therefore, it is a major disadvantage of this implementation of the ML problem.

Moreover, to achieve more satisfactory results, implementing more features into the algorithm can be viewed as an improvement for this ML method. In addition, considering the error values calculated for the model, implementing other loss functions is a possibility.

6 References

JUNG A., 2022. Machine Learning: The Basics [online]. Springer, Singapore. [viewed 8 October 2022]. Available from: https://github.com/alexjungaalto/MachineLearningTheBasics/blob/master/MLBasicsBook.pdf

StatFin, 2021. 12dx -- Vital statistics and population, 1749-2021 [online]. Statistics Finland's free-of-charge statistical databases. [viewed 28 September 2022]. Available from: https://pxweb2.stat.fi/PxWeb/pxweb/en/StatFin/StatFin__synt/statfin_synt_pxt_12dx.px/

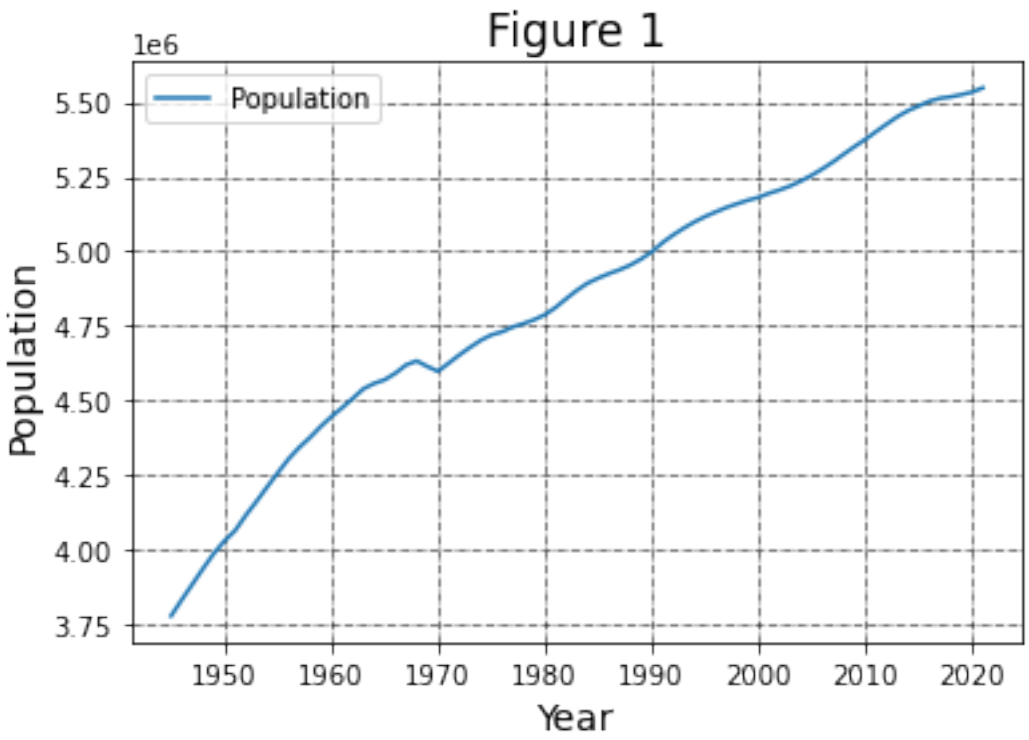
Untitled

October 10, 2022

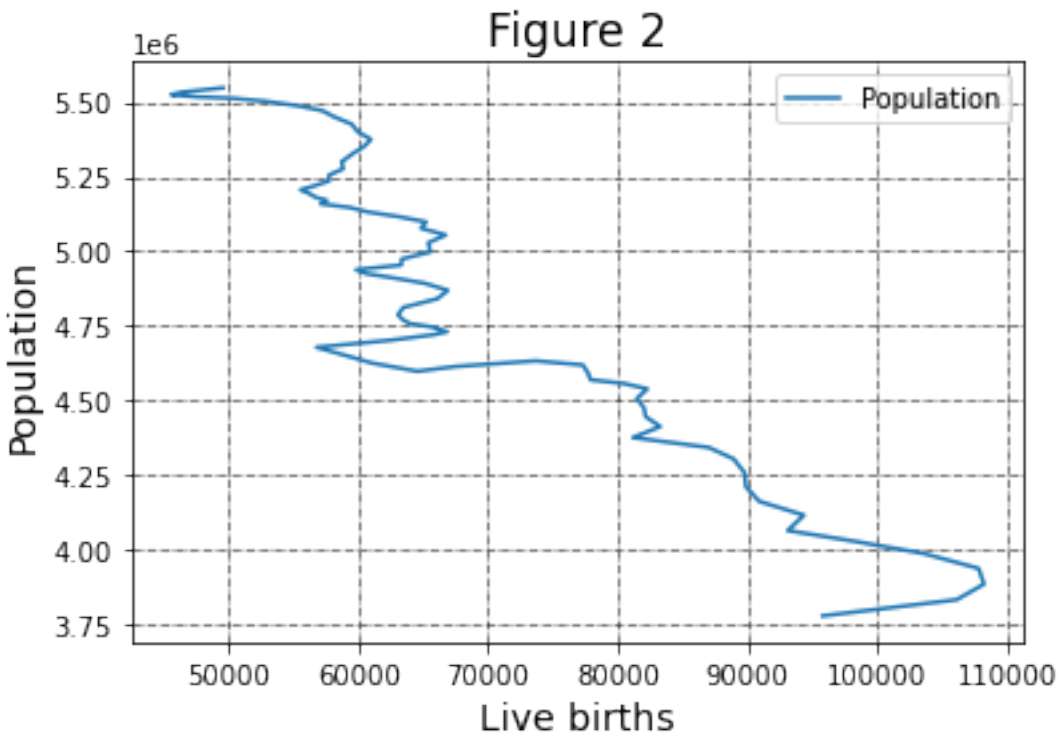
[97]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
df = pd.read_csv('Population_of_Finland.csv')
df = df.drop(['Natural increase','Intermunicipal migration','Net␣
 ↪migration','Marriages','Divorces','Total change'], axis=1)
df = df[df['Emigration from Finland'] != '.']
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 77 entries, 196 to 272
Data columns (total 6 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Year                     77 non-null     int64
 1   Live births              77 non-null     int64
 2   Deaths                   77 non-null     int64
 3   Immigration to Finland   77 non-null     object
 4   Emigration from Finland  77 non-null     object
 5   Population               77 non-null     int64
dtypes: int64(4), object(2)
memory usage: 4.2+ KB
None
```

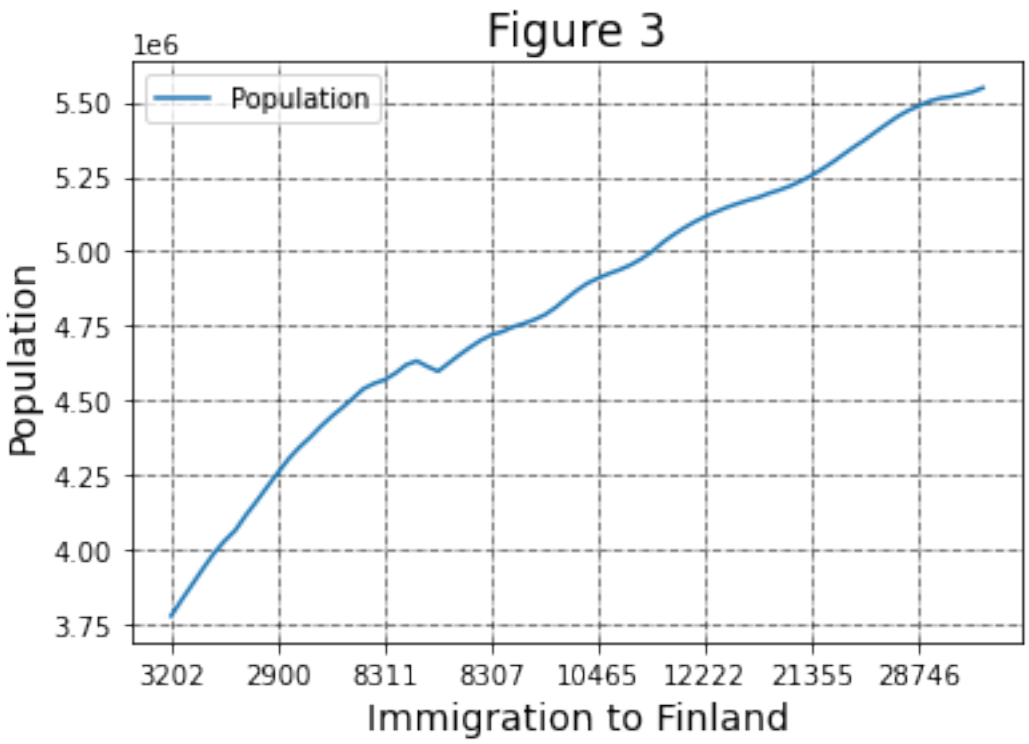
[98]:
```
df.plot('Year', 'Population').plot(figsize=(10, 7))
plt.title("Figure 1", fontsize=17)
plt.ylabel('Population', fontsize=14)
plt.xlabel('Year', fontsize=14)
plt.grid(which="major", color='k', linestyle='-.', linewidth=0.5)
plt.show()
```



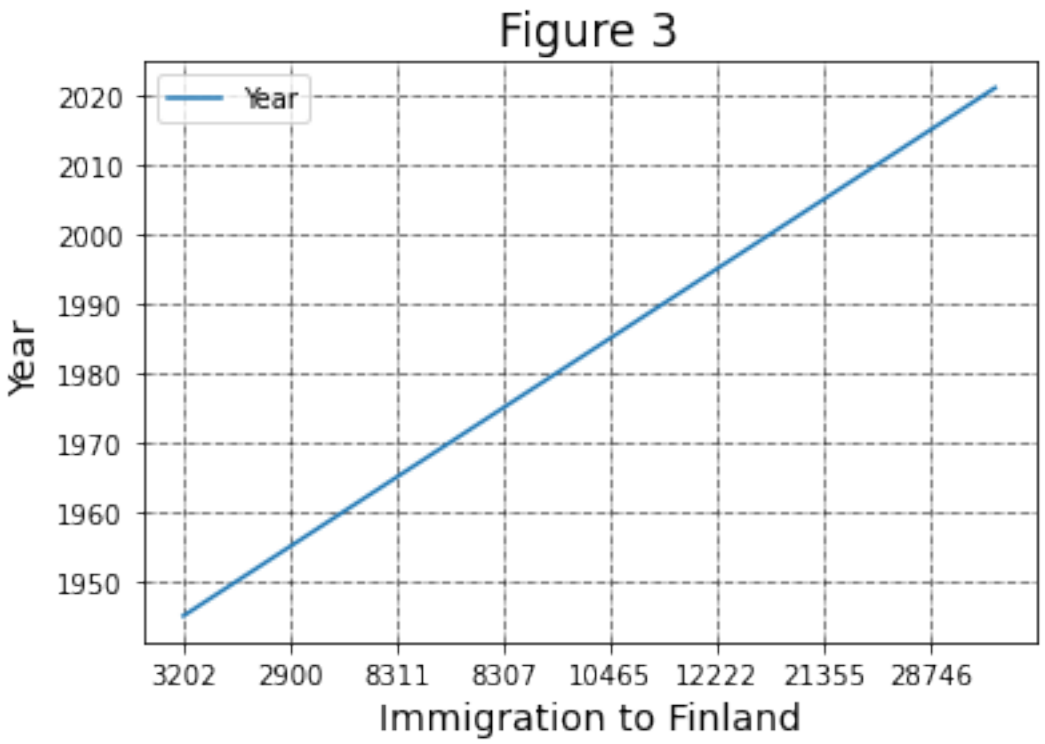
[99]:
```
df.plot('Live births', 'Population').plot(figsize=(10, 7))
plt.title("Figure 2", fontsize=17)
plt.ylabel('Population', fontsize=14)
plt.xlabel('Live births', fontsize=14)
plt.grid(which="major", color='k', linestyle='-.', linewidth=0.5)
plt.show()
```



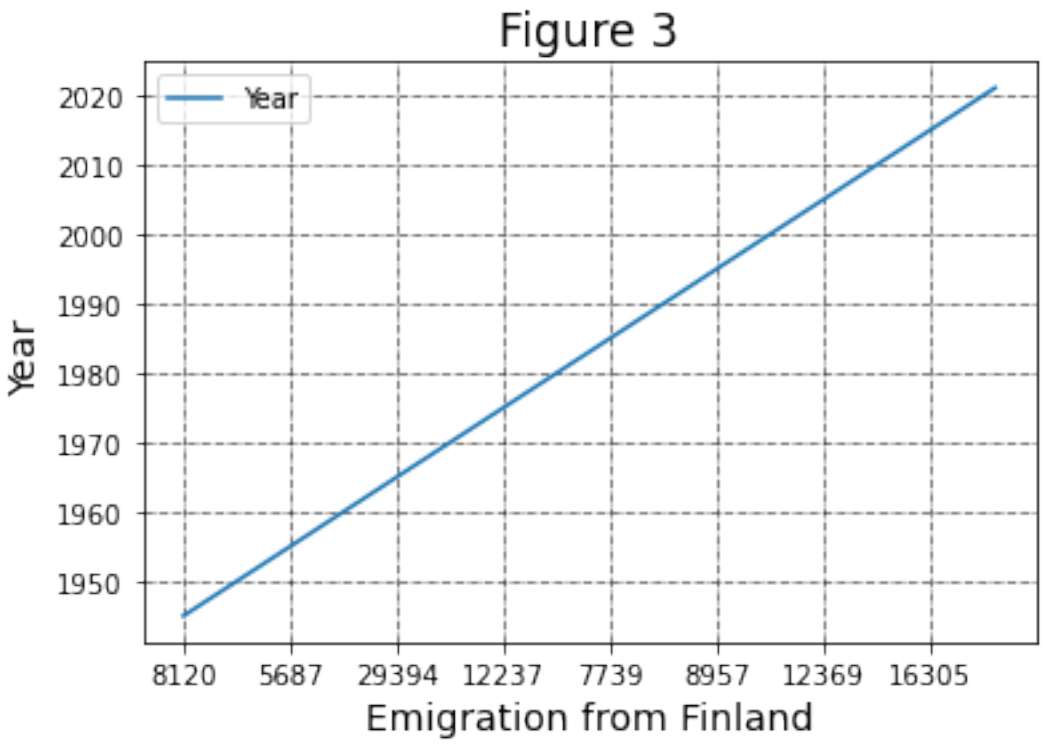
[100]:
```
df.plot('Immigration to Finland','Population').plot(figsize=(10, 7))
plt.title("Figure 3", fontsize=17)
plt.ylabel('Population', fontsize=14)
plt.xlabel('Immigration to Finland', fontsize=14)
plt.grid(which="major", color='k', linestyle='-.', linewidth=0.5)
plt.show()
```



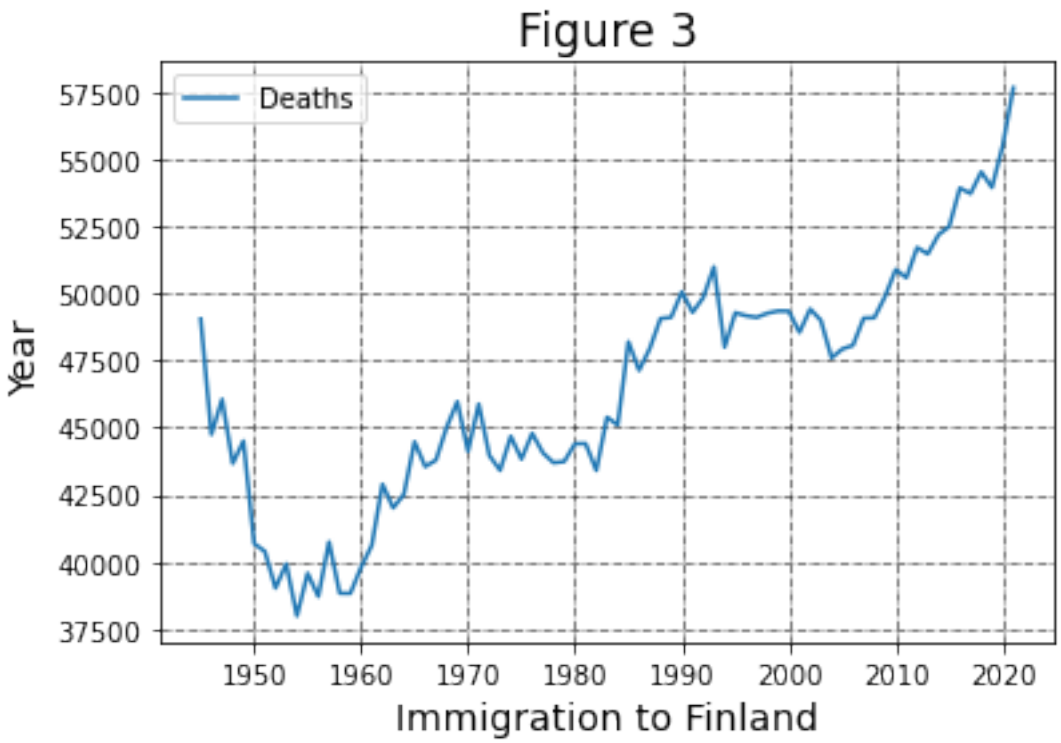
[101]:
```
df.plot('Immigration to Finland', 'Year').plot(figsize=(10, 7))
plt.title("Figure 3", fontsize=17)
plt.ylabel('Year', fontsize=14)
plt.xlabel('Immigration to Finland', fontsize=14)
plt.grid(which="major", color='k', linestyle='-.', linewidth=0.5)
plt.show()
```



[102]:
```
df.plot('Emigration from Finland', 'Year').plot(figsize=(10, 7))
plt.title("Figure 3", fontsize=17)
plt.ylabel('Year', fontsize=14)
plt.xlabel('Emigration from Finland', fontsize=14)
plt.grid(which="major", color='k', linestyle='-.', linewidth=0.5)
plt.show()
```



[103]:
```
df.plot('Year', 'Deaths').plot(figsize=(10, 7))
plt.title("Figure 3", fontsize=17)
plt.ylabel('Year', fontsize=14)
plt.xlabel('Immigration to Finland', fontsize=14)
plt.grid(which="major", color='k', linestyle='-.', linewidth=0.5)
plt.show()
```



[104]:
```
X = df[["Live births","Deaths","Immigration to Finland","Emigration from␣
 ↪Finland"]]
y = df['Population']
#X = StandardScaler().fit(X).transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.4,␣
 ↪random_state=0)
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size = 0.
 ↪5, random_state=0)
print(len(X_train),len(X_val), len(X_test))
```

```
46 16 15
```

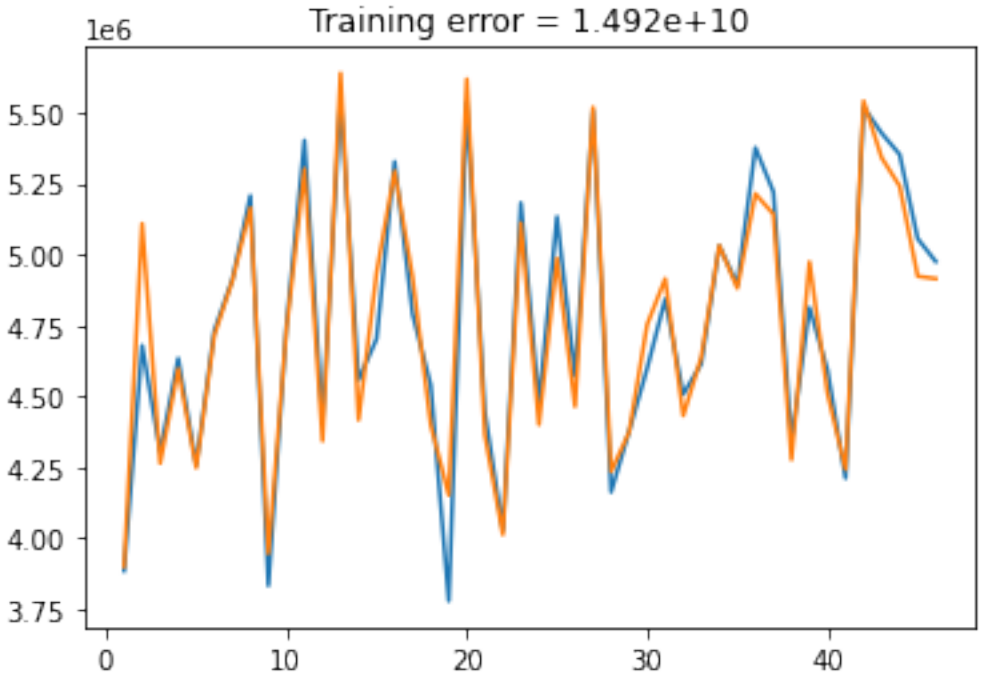
Implementing Linear Regression model:

[105]:
```
regr = LinearRegression()
regr.fit(X_train, y_train) #fitting a training set
print('Intercept: \n', regr.intercept_)
print('Coefficients: \n', regr.coef_)
y_pred = regr.predict(X_train)
tr_error = mean_squared_error(y_train, y_pred)
print('The training error is: ', tr_error)
```

```
Intercept:
 5805315.104866516
```

```
Coefficients:
 [-18.70358876   2.40672698  18.97536721  -5.08744718]
The training error is:  14920337950.932384
```

[106]:
```
fig, ax = plt.subplots()
coordinate_x=[]
for i in range(1,47):
    coordinate_x.append(i)
ax.plot(coordinate_x, y_train)
ax.plot(coordinate_x, y_pred)
plt.title('Training error = {:.5}'.format(tr_error))
plt.show()
```



[107]:
```
regr = LinearRegression()
regr.fit(X_val, y_val) #fitting a validation set
print('Intercept: \n', regr.intercept_)
print('Coefficients: \n', regr.coef_)
y_pred = regr.predict(X_val)
tr_error = mean_squared_error(y_val, y_pred)
print('The training error is: ', tr_error)
```

```
Intercept:
 2773723.2086173566
Coefficients:
 [ -2.72291266  47.73484278  23.44061535 -21.46940913]
```

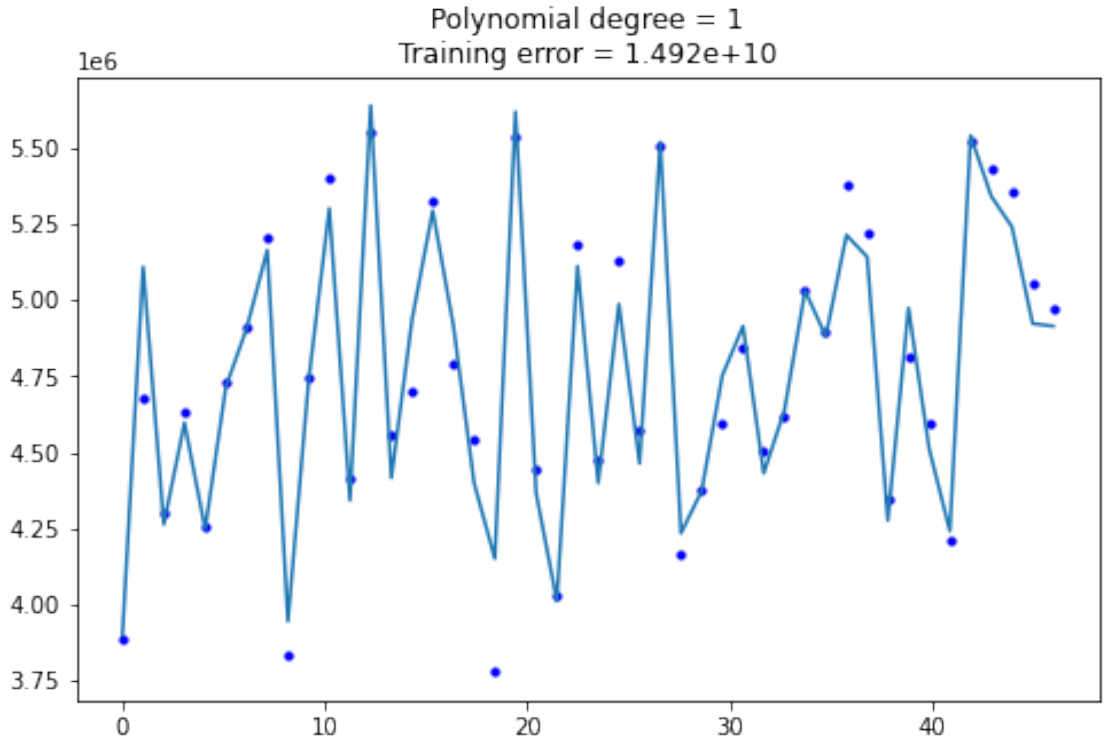
```
The training error is:  12756124728.661667
```

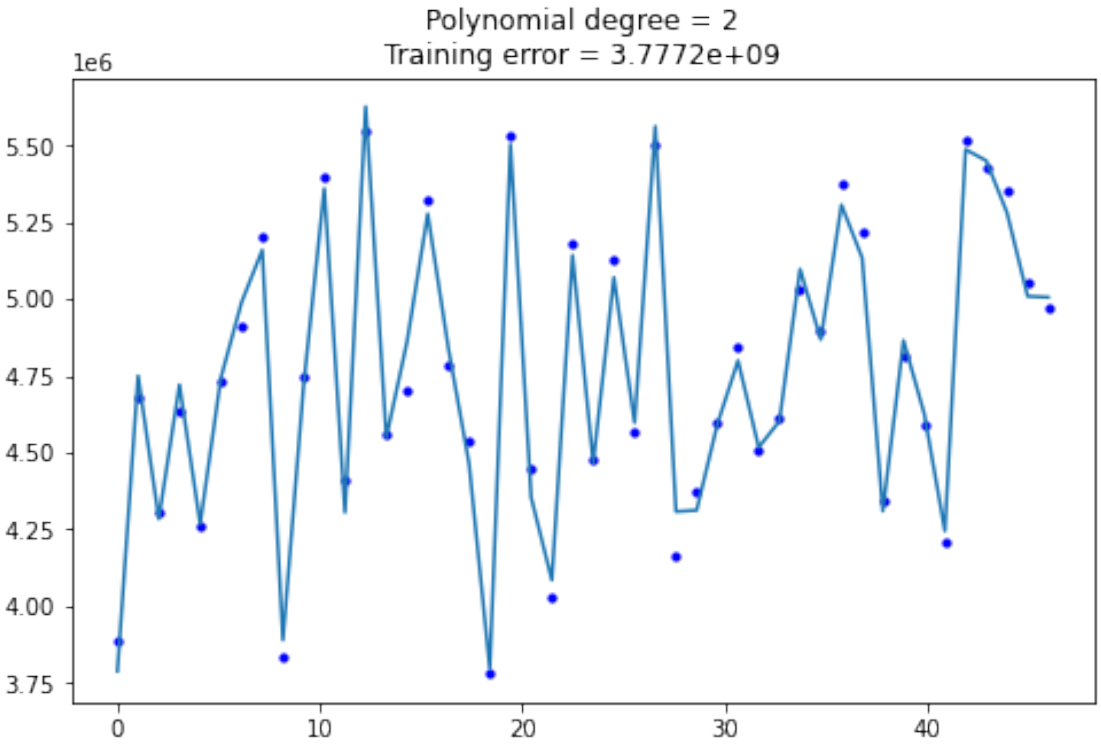
[108]:
```
fig, ax = plt.subplots()
coordinate_x = []
for i in range(16):
    coordinate_x.append(i)
ax.plot(coordinate_x, y_val)
ax.plot(coordinate_x, y_pred)
plt.title('Validation error = {:.5}'.format(tr_error))
plt.show()
```

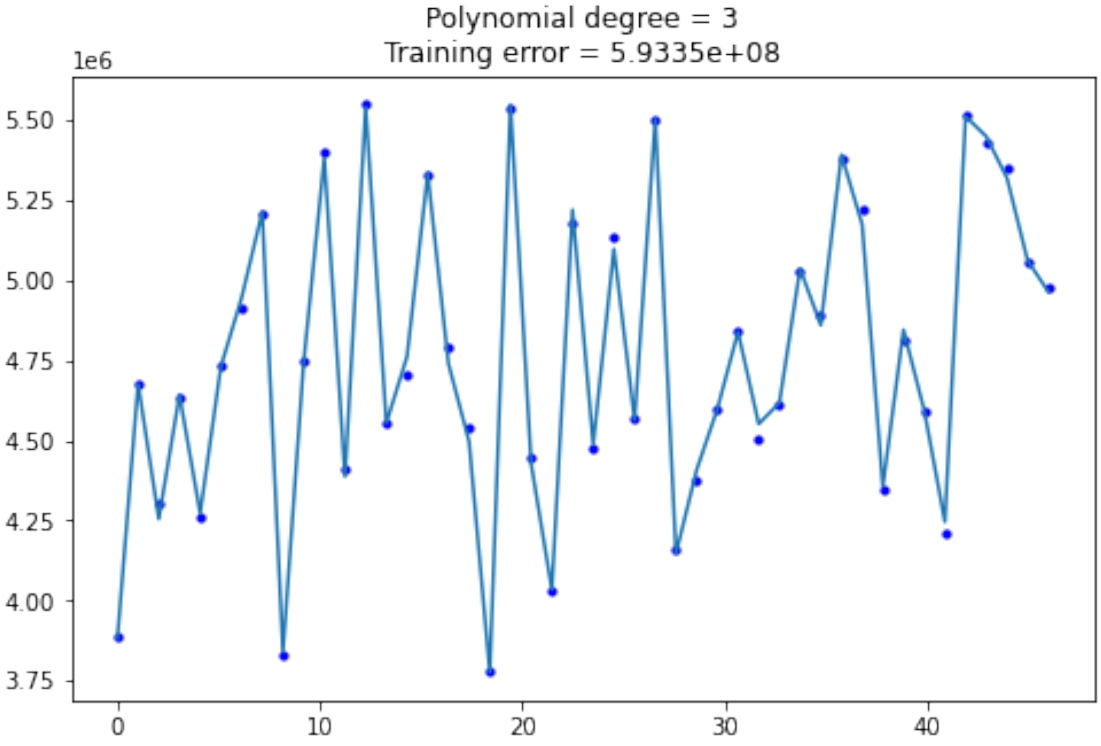
Implementing Polynomial Regression model:

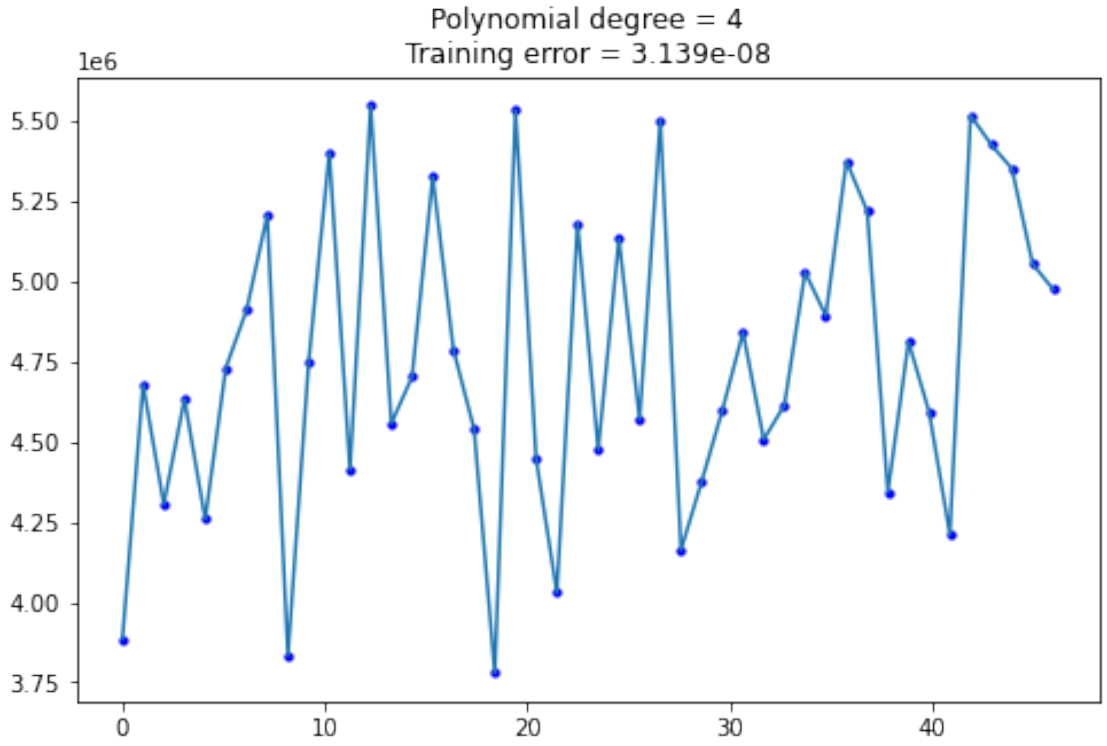
[116]:
```
degrees = [1,2,3,4] #Polynomial degrees
tr_errors = []
for n in range(len(degrees)):
    poly = PolynomialFeatures(degree = degrees[n])
    X_train_poly = poly.fit_transform(X_train) #fitting a training set
    regr = LinearRegression(fit_intercept = False)
    regr.fit(X_train_poly, y_train)
    y_pred_train = regr.predict(X_train_poly)
    tr_error = mean_squared_error(y_train,y_pred_train)
    tr_errors.append(tr_error)
    X_fit = np.linspace(0, 46, 46)
    plt.figure(figsize=(8, 5))
```

```
    plt.plot(X_fit, y_pred_train)
    plt.scatter(X_fit, y_train, color="b", s=10)
    plt.title('Polynomial degree = {}\nTraining error = {:.5}'.
 ↪format(degrees[n], tr_error))
plt.show()
print(tr_errors)
```



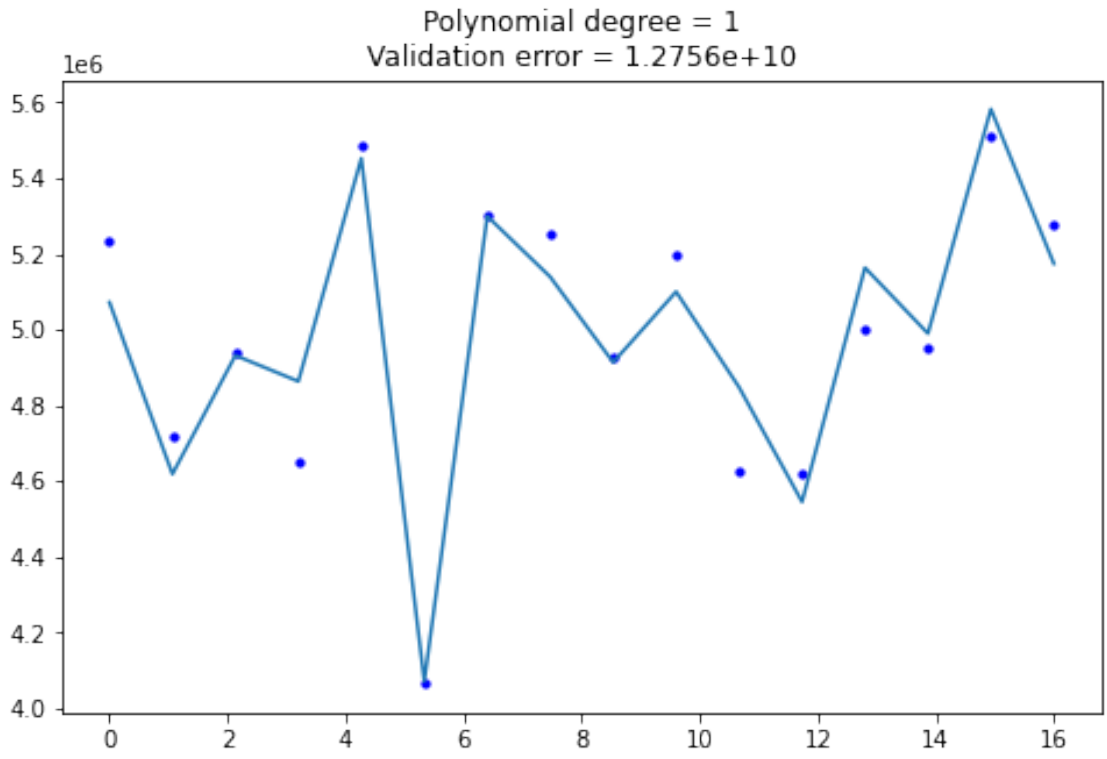

Polynomial degree = 2
Training error = 3.7772e+09
1e6
5.50
5.25
5.00
4.75
4.50
4.25
4.00
3.75
0
10
20
30
40

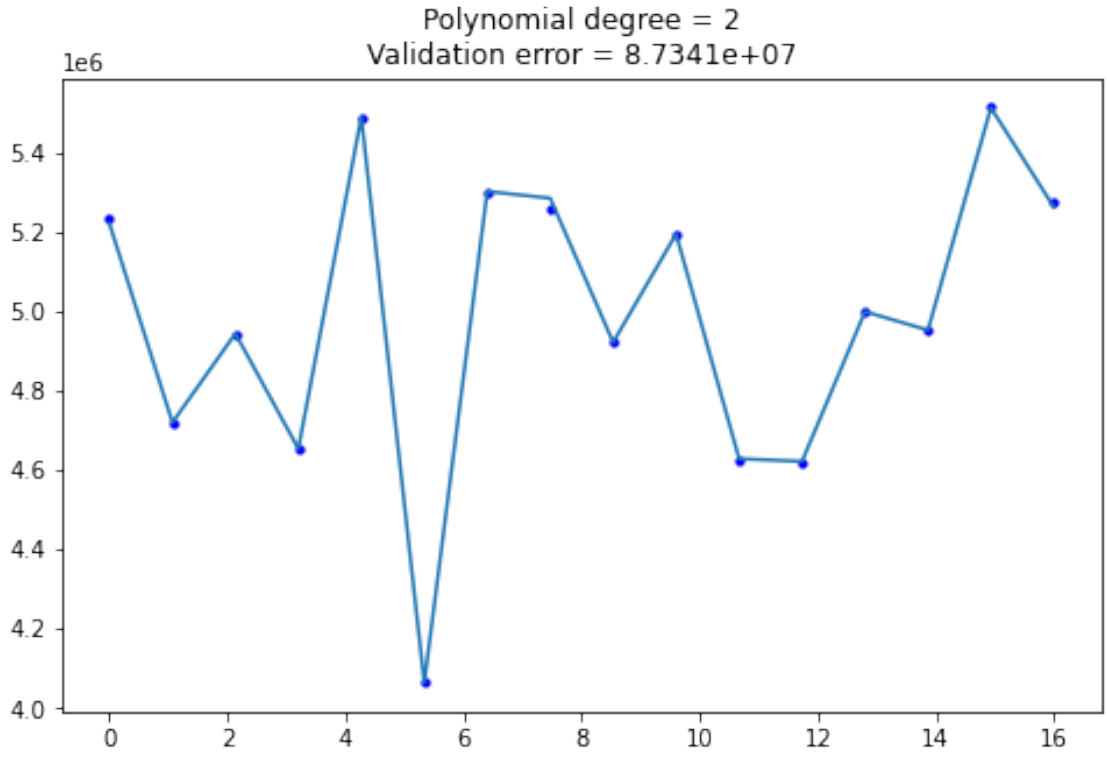

Polynomial degree = 3
Training error = 5.9335e+08
1e6
5.50
5.25
5.00
4.75
4.50
4.25
4.00
3.75
0
10
20
30
40

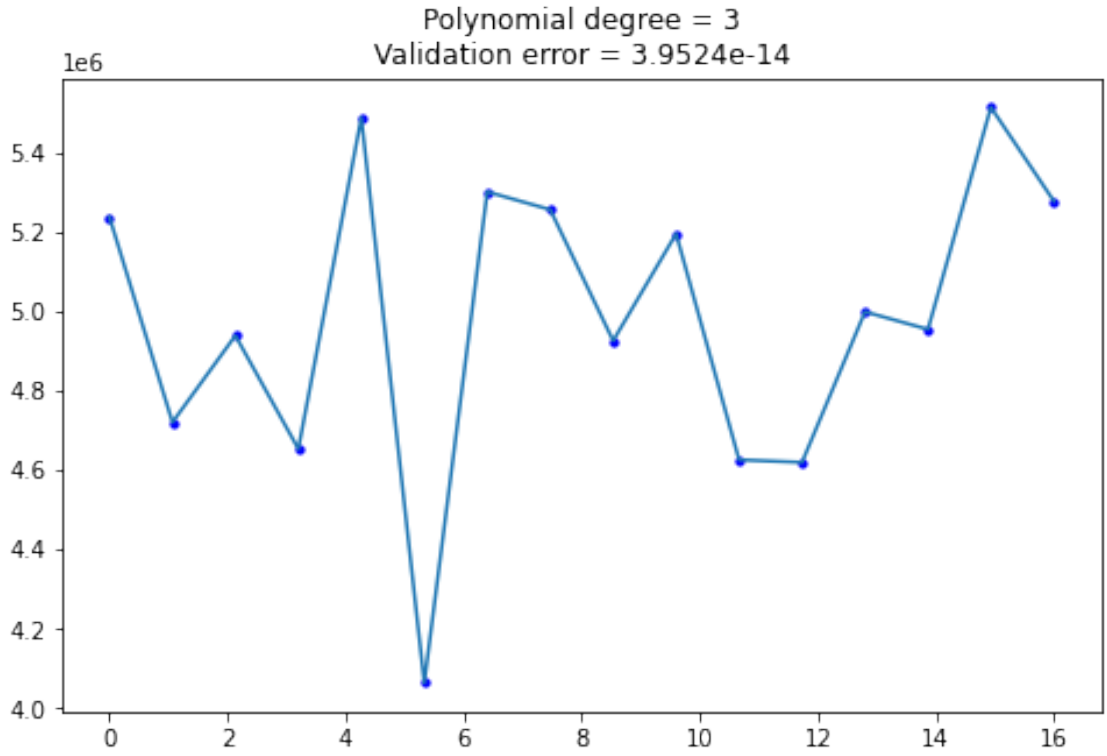


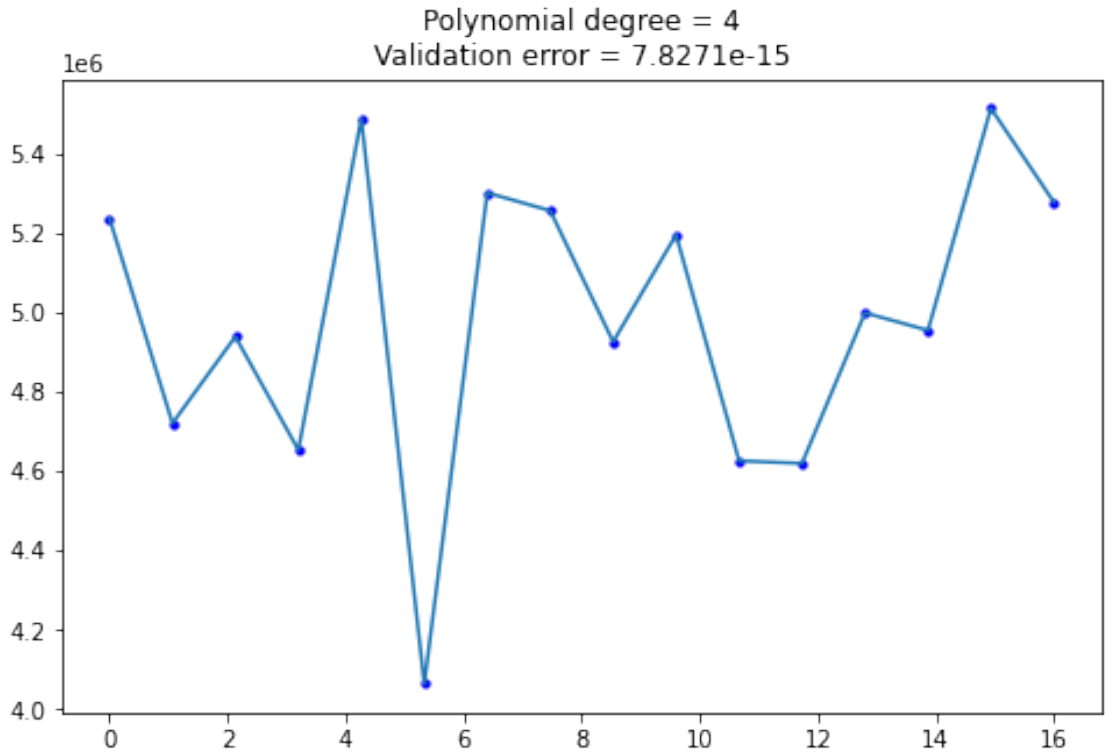

```
[14920337950.932379, 3777242758.5993376, 593354659.0069556,
3.1390133313913757e-08]
```

[117]:
```
degrees = [1,2,3,4]
val_errors = []
for n in range(len(degrees)):
    poly = PolynomialFeatures(degree = degrees[n])
    X_val_poly = poly.fit_transform(X_val) #fitting a validation set
    regr = LinearRegression(fit_intercept = False)
    regr.fit(X_val_poly, y_val)
    y_pred_val = regr.predict(X_val_poly)
    val_error = mean_squared_error(y_val, y_pred_val)
    val_errors.append(val_error)
    X_fit = np.linspace(0, 16, 16)
    plt.figure(figsize=(8, 5))
    plt.plot(X_fit, y_pred_val)
    plt.scatter(X_fit, y_val, color="b", s=10)
    plt.title('Polynomial degree = {}\nValidation error = {:.5}'.
 ↪format(degrees[n], val_error))
plt.show()
print(val_errors)
```


Polynomial degree = 1
Validation error = 1.2756e+10
1e6
5.6
5.4
5.2
5.0
4.8
4.6
4.4
4.2
4.0
0
2
4
6
8
10
12
14
16

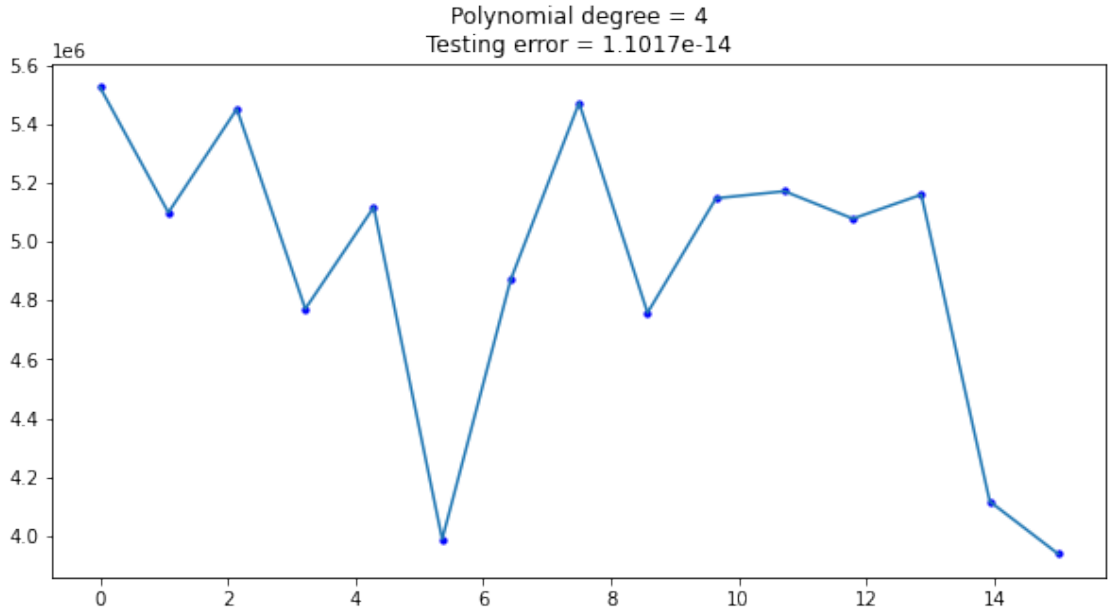

Polynomial degree = 2
Validation error = 8.7341e+07
1e6
5.4
5.2
5.0
4.8
4.6
4.4
4.2
4.0
0
2
4
6
8
10
12
14
16


Polynomial degree = 3
Validation error = 3.9524e-14
1e6
5.4
5.2
5.0
4.8
4.6
4.4
4.2
4.0
0
2
4
6
8
10
12
14
16


Polynomial degree = 4
Validation error = 7.8271e-15
1e6
5.4
5.2
5.0
4.8
4.6
4.4
4.2
4.0
0
2
4
6
8
10
12
14
16

```
[12756124728.661676, 87341325.65646063, 3.952393967665557e-14,
7.827072323607354e-15]
```

[124]:
```
degrees = [4]
test_errors = []
for n in range(len(degrees)):
    polyx = PolynomialFeatures(degree = degrees[n])
    X_test_poly = poly.fit_transform(X_test) #fitting a test set
    regr = LinearRegression(fit_intercept = False)
    regr.fit(X_test_poly, y_test)
    y_pred_test = regr.predict(X_test_poly)
    test_error = mean_squared_error(y_test, y_pred_test)
    test_errors.append(test_error)
    X_fit = np.linspace(0, 15, 15)
    plt.figure(figsize=(10, 5))
    plt.plot(X_fit, y_pred_test)
    plt.scatter(X_fit, y_test, color="b", s=10)
    plt.title('Polynomial degree = {}\nTesting error = {:.5}'.
 ↪format(degrees[n], test_error))
plt.show()
print(test_errors)
```




```
[1.101711314769697e-14]
```

[115]:
```
#poly_pred = regr2.predict(polyx.fit_transform([[2025]]))
#print(poly_pred)
from sklearn.metrics import r2_score
r2_score(y_test, y_pred_test)
```

[115]: 1.0

[ ]:
```

```