Polynomial Regression and Linear Regression to determine in-game users' activity.

1. Introduction:

In the past 20 years, Steam has exploded as a platform for people to buy, play and share their love for games. As a person that also love and enjoy playing games, I want to know how many of the concurrent users are playing in a game to see how active the user base is on a particular day. The benefit of this is to determine whether the traffic on that day is stable or not, as spikes in user count may affect the server.

This section briefly introduces the aim and application of this project

+ Section 2 will explain the problem formulation in more details, including data points, features, and labels, and data sources.

+ Section 3 discusses the methods for this problem, including models, loss functions and splitting of dataset for model validation.

+ Section 4 presents the results about the performance of the mentioned methods

+ Section 5 concludes the report with summary, result interpretation and some future directions.
+ Section 6 are a list of references.
+ Section 7 is an appendix where you can see the codes for the project.

2. Problem Formulation

The aim of this project is to predict the total in-game users from the total online users of Steam on a specific Date.

Therefore, the datapoints in this project are days, each has different concurrent users count and concurrent in-game user count. The feature is the concurrent users. The label is the concurrent in-game users.

Summary:

- **Aim of project:** predict the concurrent in-game users on steam based on the concurrent users count

- **Data point:** a day

- **Feature(s):** Concurrent users on Steam (numerical – float)

- **Label(s):** Concurrent in-game users (numerical - float)

3. Method

Dataset

By sorting the data gathered from SteamDB [1] and Steam's own Steam & Game Stats page [2], I have created the current data set, which can be further updated as the data are gathered daily on the mentioned websites.

This dataset currently consists of 799 data points, each contains info of the date the data was gathered, the concurrent users and the concurrent in-game users of that day. The data set will be split into a test set (10%) and the remaining set (90%) for training and validating. This proportion enables enough data to be used for training and model selection. A test set has also been created to evaluate overall performance and avoid overfitting to validation error. I will be using k-fold validation to produce more accurate errors estimation because it is applicable in this set, and is more efficient when compare with single split [3]

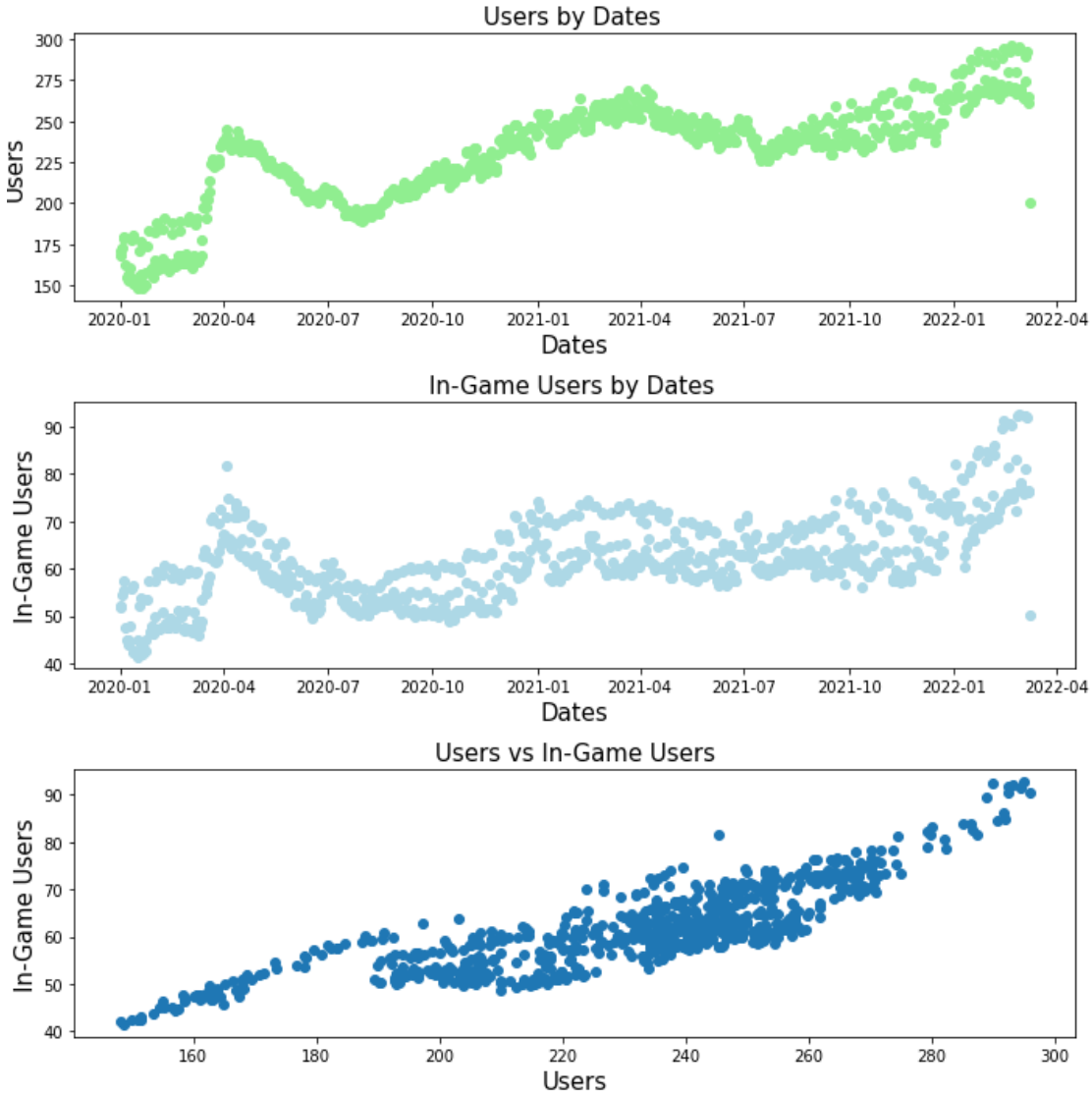
For K-Fold validation, the dataset (remaining set) will be divided and folded with k = 7 (7 subsets, folded 7 times) for Linear Regression Model and k = 3 for Polynomial Regression Model (of degrees 3,5,10).

The data set consist of 3 column "Dates", "Users" and "In-game" where 2 columns are possible candidates to act as feature.

Feature selection

The data points consist of the dates, the concurrent users, and concurrent in-game users. It easy to see the correlation between the two "users" column, if one is increasing, the other has a high chance of also increasing, while nothing can be gathered from the dates themselves.

By plotting the data points, the correlation between the two can be seen more clearly. Thus, the feature is the data from the "Users" column



(The numbers are in million)

Model selection

Model 1: Linear Regression

As seen in the plots, the concurrent users show a linear correlation with the concurrent in-game users. Hence, a linear method is suitable for this ML problem and thus, the linear regression method is chosen.
The label can be calculated using the formula:

$$\hat{y} = \alpha_1 x + \alpha_0$$

With y being the label (concurrent in-game users), x being the feature (concurrent users).

The mean squared error is calculated by the formula:

$$\text{MSE} = \frac{1}{k}\sum_{i=1}^{k}(y_i - \hat{y}_i)^2$$

Where k is the number of data points, y_i is the label and $\hat{y}_i$ the predicted label based on feature of the i^{th} data point. This loss function is chosen because it is often used in linear regression model [3], it is also supported by the sci-kit learn package with the function mean_squared_error which helps with the ease of use [4].

Model 2: Polynomial Regression

Because of the linear relationship shown in the plot, another suitable linear method is polynomial regression.

The label can be calculated using this formula for the 1st degree:

$$\hat{y} = mx + b$$

Where y is the label (concurrent in-game users), x being the feature (concurrent users). As degree of the model increases, the formula will change so here I only provide an example for the 1st degree.

Different orders of polynomials (d = 2 . . . 10) are fitted to the training set by minimizing the mean squared loss. Mean squared loss is calculated by summing over all data points and obtaining the squared difference for the predicted value given hypothesis, here being polynomial (This also make use of the same mean squared error from above). This loss function is chosen because it also make use of the mean squared_error function from the sci-lit learn package [4].

4. Result

For the 1st Model (Linear regression):
The MSE obtained from the 7 CV-iterations fluctuate quite considerably, with an average validation error of 18.262265533248392 and an average training error of 18.193783108191226.

For the 2nd Model (Polynomial regression):
The result obtained from the 3 CV-iterations, with the lowest training and validation errors is of degree 10. This model has an average validation error of 15.9888463004725452 and an average training error of 15.7764456997498641

From the 2 models, we can see that Polynomial regression should be chosen because of the lower erros, thus the final model is Polynomial Regression of degree 10.

For the Final Model (Polynomial regression):

The test set is created by splitting the original dataset into 10% for testing and 90% for using in K-Fold CV (described in Section 3).

The final chosen model is obtained by fitting polynomial regression to remaining set for K-Fold CV and then tested using the test set. The training error of the final model is 15.549057442758352 with testing error 14.388303242207702.

5. Conclusion

The report discusses the performance of linear regression and polynomial regression models for this ML problem. The mean squared error is chosen as the loss function, and K-Fold cross validation is used for model selection.

+ Linear regression model has average training (18.262265533248392) and validation (18.193783108191226) errors which are relatively close to each other.

+ Similarly, polynomial regression model with the lowest average validation error (polynomial of degree 10) also has average validation errors and training errors that are close to each other (15.9888463004725452 and 15.7764456997498641 respectively). The results here are lower than those from linear regression model.

The model chosen is polynomial regression because of the lower training and validation errors, and the final model has a test error of 14.388303242207702 and a training error of 15.549057442758352. It noticeable that both the errors are relatively low and close to each other, and the test error is also lower than the training error, which suggests that the model is likely to perform well on unseen data.

To improve the methods for this problem. We can:
+ Use and compare the results with other models, for example decision tree because of the outliers in the data set.
+ Another the way is to add additional features that considers holidays or events.
All of these will then require more data to obtain optimal result.

6. References

[1] SteamDB: https://steamdb.info/app/753/graphs/

[2] Steam & Game Stats: https://store.steampowered.com/stats/Steam-Game-and-Player-Statistics

[3] Coursebook, Chapter 2.3

[4] Scikit Learn, https://scikit-learn.org/stable/index.html

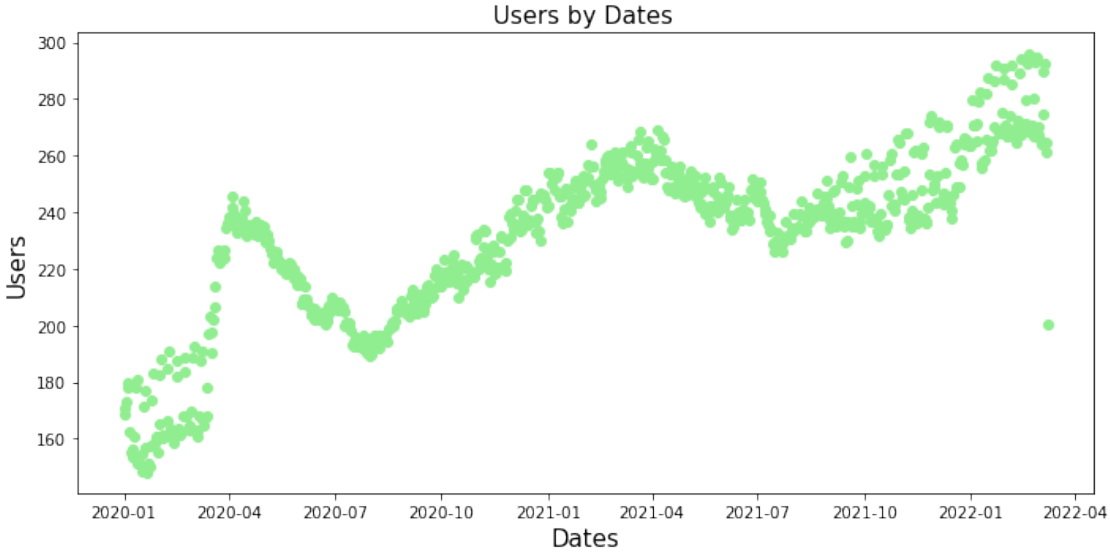
[5] Dataset Link: https://docs.google.com/spreadsheets/d/1OiPkJ-U3aKz5OXe9rUhGEUvOXE0WmUNTGHP8Nu9j4Dk/edit?usp=sharing

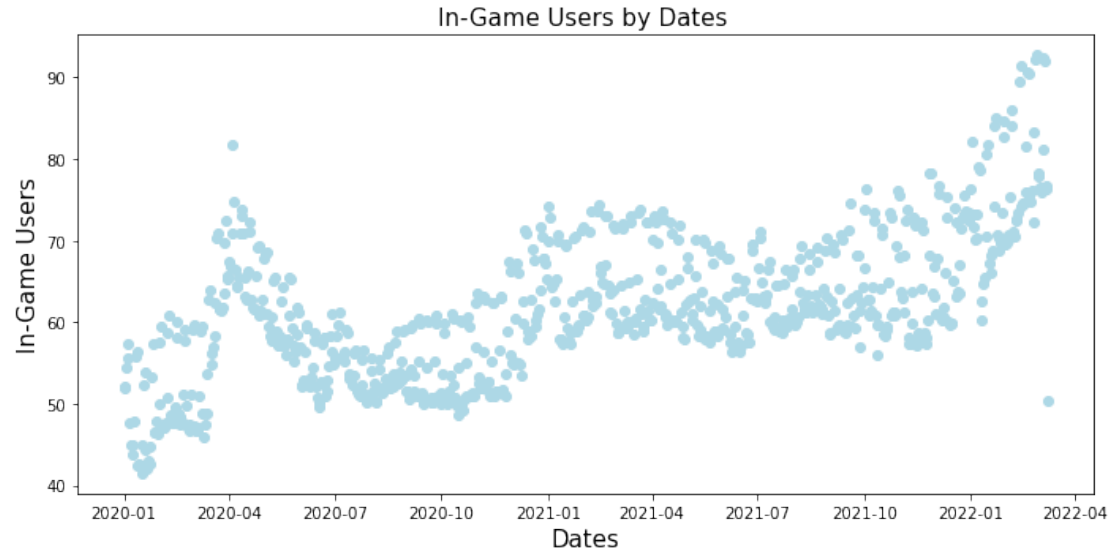
7. Appendix

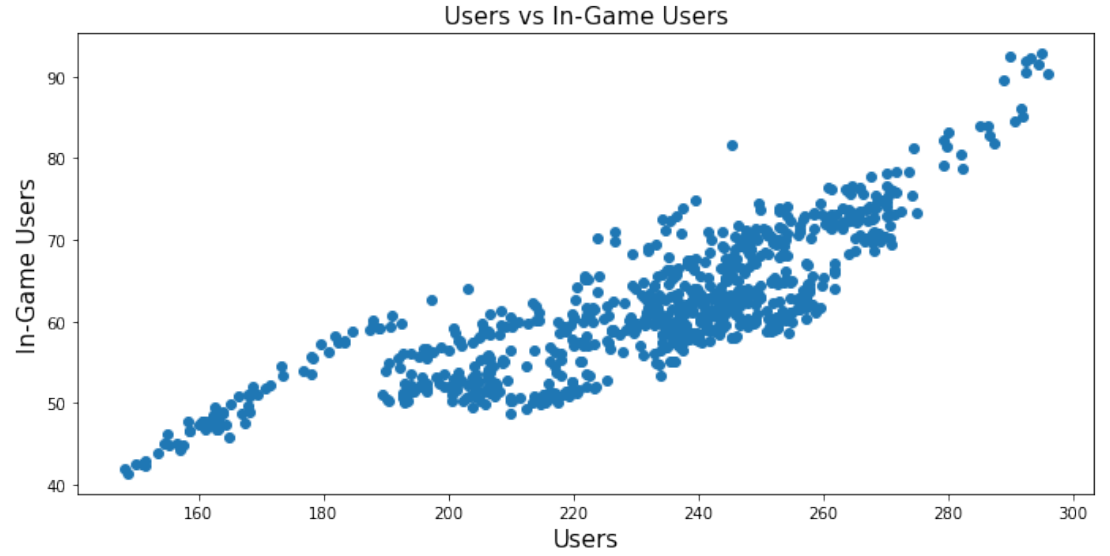
Codes

March 30, 2022

[185]:
```
import pandas as pd
import matplotlib.pyplot as plt
SteamRaw = pd.read_excel("Steam.xlsx")
SteamRaw['Dates'] = pd.to_datetime(SteamRaw['Date']).dt.date
SteamRefined = SteamRaw.drop(columns=['Date'])
SteamRefined = SteamRefined[['Dates', 'Users', 'In-Game']]
SteamRefined['Users'] = SteamRefined['Users'].div(100000).round(2)
SteamRefined['In-Game'] = SteamRefined['In-Game'].div(100000).round(2)
SteamRefined.to_csv('Steam.csv')

fig, axes = plt.subplots(3, figsize=(10,15))
axes[0].scatter(SteamRefined['Dates'],SteamRefined['Users'],color =␣
↪'lightgreen')
axes[0].set_xlabel("Dates",size=15)
axes[0].set_ylabel("Users",size=15)
axes[0].set_title("Users by Dates ",size=15)

axes[1].scatter(SteamRefined['Dates'],SteamRefined['In-Game'],color =␣
↪'lightblue')
axes[1].set_xlabel("Dates",size=15)
axes[1].set_ylabel("In-Game Users",size=15)
axes[1].set_title("In-Game Users by Dates ",size=15)

axes[2].scatter(SteamRefined['Users'],SteamRefined['In-Game'])
axes[2].set_xlabel("Users",size=15)
axes[2].set_ylabel("In-Game Users",size=15)
axes[2].set_title("Users vs In-Game Users",size=15)
fig.tight_layout()
plt.show()
```


Users by Dates
Users
Dates
300
280
260
240
220
200
180
160
2020-01
2020-04
2020-07
2020-10
2021-01
2021-04
2021-07
2021-10
2022-01
2022-04


In-Game Users by Dates
In-Game Users
Dates
90
80
70
60
50
40
2020-01
2020-04
2020-07
2020-10
2021-01
2021-04
2021-07
2021-10
2022-01
2022-04


Users vs In-Game Users
In-Game Users
Users
90
80
70
60
50
40
160
180
200
220
240
260
280
300

[186]:
```
import numpy as np
import pandas as pd
from statistics import mean
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures

SteamCurrent = pd.read_csv('Steam.csv')
SteamCurrent = SteamCurrent[['Dates','Users','In-Game']]

tr_errors = []
val_errors = []
coefficients = []
intercepts = []
x = 0

X=SteamCurrent['Users'].to_numpy().reshape(-1,1)
y=SteamCurrent['In-Game'].to_numpy()
X_rem, X_test, y_rem, y_test = train_test_split(X,y,test_size=0.1,random_state␣
 ↪= 42)
k = KFold(n_splits=7,shuffle=True,random_state=42)

print("\nFirst Model: Linear Regression\n")

for (i, j) in k.split(X_rem):
    X_train, y_train, X_val, y_val = X[i], y[i],X[j], y[j]
    lin_regr = LinearRegression().fit(X_train, y_train)
    y_pred_train = lin_regr.predict(X_train)
    y_pred_val = lin_regr.predict(X_val)
    tr_error = mean_squared_error(y_train, y_pred_train)
    val_error = mean_squared_error(y_val, y_pred_val)
    val_errors.append(val_error)
    tr_errors.append(tr_error)
    intercepts.append(lin_regr.intercept_)
    coefficients.append(lin_regr.coef_)
    x += 1

    print("Iteration No.: " + str(x))
    print("Training error: " + str(tr_error))
    print("Validation error: " + str(val_error))
    print("\n")

print("Average val error: " + str(mean(val_errors)))
```

```
print("Average tr error: " + str(mean(tr_errors)))
print("\nLearned intercepts: " + str(intercepts))
print("Learned weights: "+ str(coefficients))
```

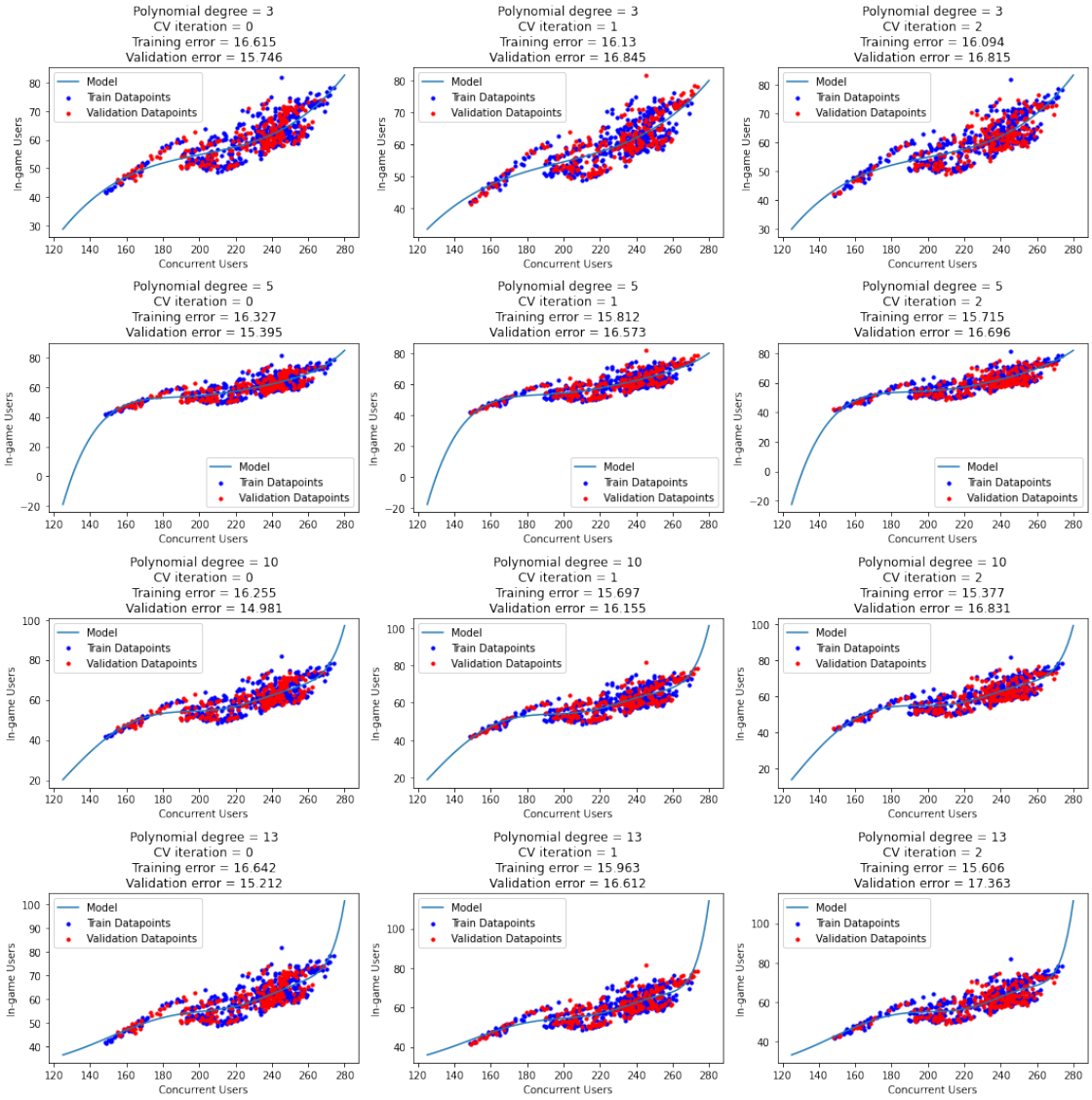
```
First Model: Linear Regression

Iteration No.: 1
Training error: 18.12136469929807
Validation error: 18.701333596997816


Iteration No.: 2
Training error: 18.731361680499916
Validation error: 15.057447698227476


Iteration No.: 3
Training error: 17.99467985064364
Validation error: 19.419520495847735


Iteration No.: 4
Training error: 17.666729958943982
Validation error: 21.406686862815693


Iteration No.: 5
Training error: 18.385629724781978
Validation error: 17.115663036611032


Iteration No.: 6
Training error: 18.112351897782347
Validation error: 18.73928144655707


Iteration No.: 7
Training error: 18.34436394538865
Validation error: 17.395925595681934


Average val error: 18.262265533248392
Average tr error: 18.193783108191226

Learned intercepts: [11.60906303302064, 11.302994074017697, 11.868319789673166,
12.305698731247425, 11.70382225615753, 12.229363132354322, 11.611424690741288]
```

```
Learned weights: [array([0.21343814]), array([0.21531366]), array([0.21262445]),
array([0.21065858]), array([0.21303664]), array([0.21104361]),
array([0.2141925])]
```

[187]:
```
SteamCurrent = pd.read_csv('Steam.csv')
SteamCurrent = SteamCurrent[['Dates','Users','In-Game']]

X=SteamCurrent['Users'].to_numpy().reshape(-1,1)
y=SteamCurrent['In-Game'].to_numpy()
X_rem, X_test, y_rem, y_test = train_test_split(X,y,test_size=0.1,random_state␣
↪= 42)
k, shuffle, seed = 3, True, 42
kFold = KFold(n_splits=k, shuffle=shuffle, random_state=seed)

degrees = [3, 5, 10, 13]

tr_errors = {}
val_errors = {}

print("\nSecond Model: Polynomial Regression")

plt.figure(figsize=(15, 15))
for i, degree in enumerate(degrees):
    tr_errors[degree] = []
    val_errors[degree] = []


    for j, (train_indices, val_indices) in enumerate(kFold.split(X_rem)):
        plt.subplot(len(degrees), k, i * k + j + 1)


        X_train, y_train, X_val, y_val = X[train_indices], y[train_indices],␣
↪X[val_indices], y[val_indices]

        lin_regr = LinearRegression()
        poly = PolynomialFeatures(degree=degree)
        X_train_poly = poly.fit_transform(X_train)
        lin_regr.fit(X_train_poly, y_train)

        y_pred_train = lin_regr.predict(X_train_poly)
        tr_error = mean_squared_error(y_train, y_pred_train)
        X_val_poly = poly.transform(X_val)
        y_pred_val = lin_regr.predict(X_val_poly)
        val_error = mean_squared_error(y_val, y_pred_val)

        tr_errors[degree].append(tr_error)
        val_errors[degree].append(val_error)
```

```
        X_fit = np.linspace(125, 280, 100)
        plt.tight_layout()
        plt.plot(X_fit, lin_regr.predict(poly.transform(X_fit.reshape(-1, 1))),␣
↪label="Model")
        plt.scatter(X_train, y_train, color="b", s=10, label="Train Datapoints")
        plt.scatter(X_val, y_val, color="r", s=10, label="Validation␣
↪Datapoints")
        plt.xlabel('Concurrent Users')
        plt.ylabel('In-game Users')
        plt.legend(loc="best")
        plt.title(f'Polynomial degree = {degree}\nCV iteration = {j}\nTraining␣
↪error = {tr_error:.5}\nValidation error = {val_error:.5}')    # set the title
plt.show()
```

Second Model: Polynomial Regression



[188]:
```
tr_errors, val_errors
print("\nResult of Second Model:\n ")
average_train_error, average_val_error = {}, {}
for degree in degrees:

    average_train_error[degree] = np.mean(tr_errors[degree])
    average_val_error[degree] = np.mean(val_errors[degree])

    print(f"Degree {degree}, Average train error = {average_train_error[degree]:
↪.16f}, "
          f"Average val error = {average_val_error[degree]:.16f}")
```

```
Result of Second Model:

Degree 3, Average train error = 16.2797841406140300, Average val error =
16.4684999258153830
Degree 5, Average train error = 15.9514848903072561, Average val error =
16.2210628678950037
Degree 10, Average train error = 15.7764456997498641, Average val error =
15.9888463004725452
Degree 13, Average train error = 16.0702690180625858, Average val error =
16.3953608005921474
```

[189]:
```
print("\nChosen Model: ")
lin_regr = LinearRegression()
poly = PolynomialFeatures(degree=10)
X_rem_poly = poly.fit_transform(X_rem)
lin_regr.fit(X_rem_poly, y_rem)
y_pred_rem = lin_regr.predict(X_rem_poly)
tr_error = mean_squared_error(y_rem,y_pred_rem)
print("\nTraining error: " + str(tr_error))

X_test_poly = poly.transform(X_test)
lin_regr.fit(X_test_poly,y_test)
y_pred_test = lin_regr.predict(X_test_poly)
test_error = mean_squared_error(y_test, y_pred_test)
print("Testing error: " + str(test_error))
```

```
Chosen Model:

Training error: 15.549057442758352
Testing error: 14.388303242207702
```

[ ]:
```

```