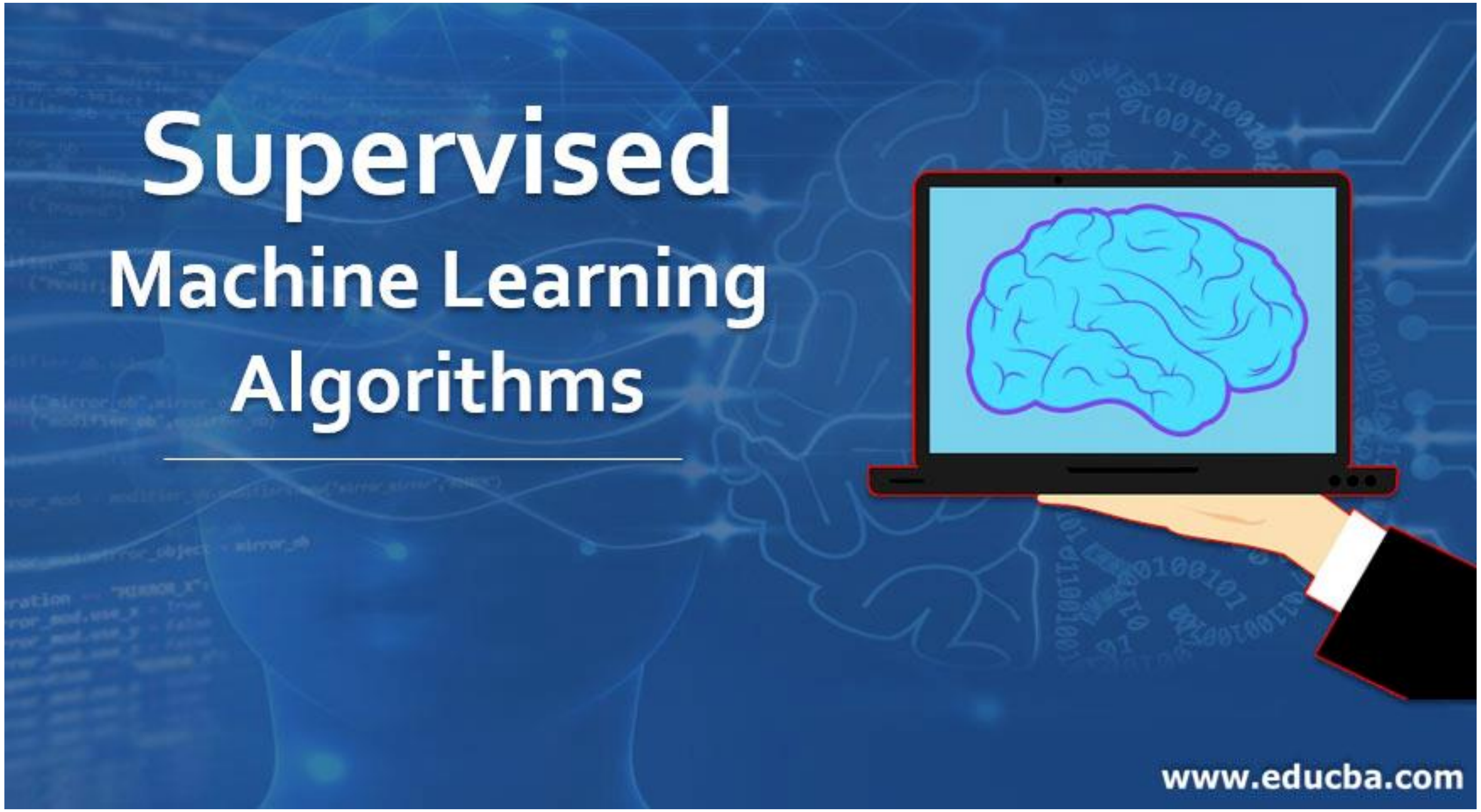


ECOM SCHOOL
המכללה למקצועות הדיגיטל וההייטק

Last lecture reminder

We learned about:

- Evaluating regression
- Error metrics (MAE, MSE, RMSE)
- Residual investigation
- Model deployment

Polynomial Regression Introduction

Polynomial regression → Polynomial regression is a form of regression analysis, which is a technique used to model the relationship between a dependent variable and one or more independent variables. Unlike linear regression, in polynomial regression, the relationship is modeled as an nth degree polynomial.

In other words, polynomial regression adds terms with degrees greater than one to the model, which allows the model to fit the data more accurately. This approach is particularly useful when the relationship
between variables is not linear, or when there are large amounts of variation in the data.

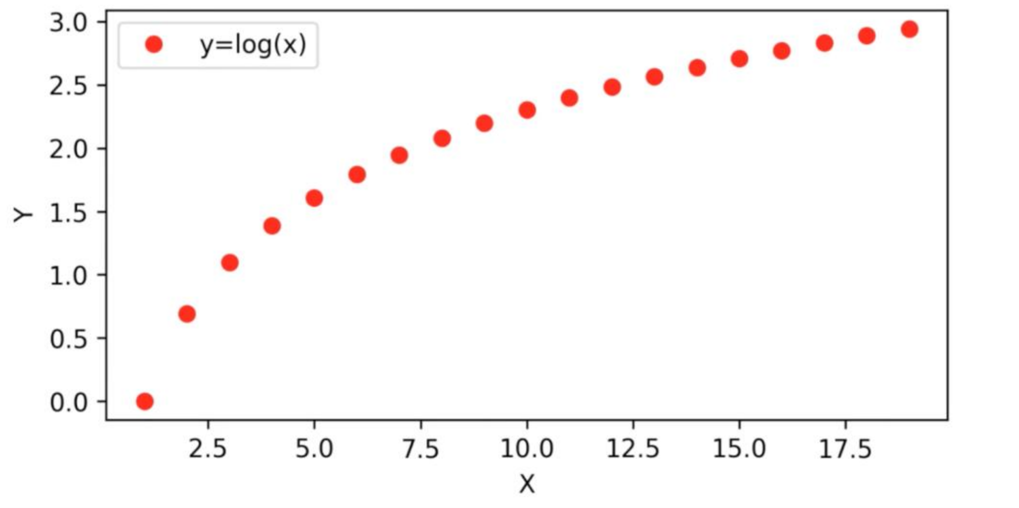
The general equation fo

$$y = b_0 + b_1 x_1 + b_2 x_1^2 + \dots + b_n x_1^n$$

ECOM SCHOOL
המכללה למקצועות הדיגיטל וההייטק

Polynomial Regression - Example

For example → Let’s take the following non linear relationship between feature x and label y

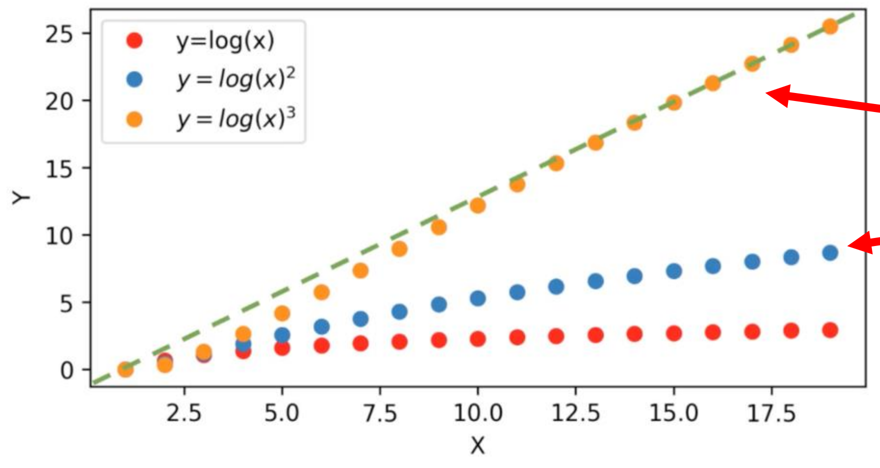


As we can see from the graph the relationship between the feature x and label y is not linear relationship (log(x) behaviour is non linear). By applying polynomial regression we could still predict based on linear regression modeling even though the relationship is not linear.

ECOM SCHOOL
המכללה למקצועות הדיגיטל וההייטק

Polynomial Regression - Example

Let's take a look what happen when we look at the graph but with higher order:



The higher the order, the more our correlation behave similar to linear correlation

Note: Not every high order feature will behave as linear correlation. This was just an example to show what is the reason behind checking for polynomial regression as well as the 1 degree linear regression.

In addition, polynomial regression can be benefit in case we have **interaction terms** between different features.

ECOM SCHOOL
המכללה למקצועות הדיגיטל וההייטק

Polynomial Regression - Feature Interaction

To better understand this point, let’s use our campaign example from previous lecture.
We saw that the beta coefficient of the newspaper expense were very close to 0 indicate that this feature is not correlated with the sales label.

What we didn’t check is that maybe there is an interaction terms between the newspaper feature and another feature, causing the newspaper to only be significant when in sync with another feature.

In order to test this interaction we can use polynomial regression to test all possible combinations between all the different features in our model.
When using, we will add new features to our model that include:

- The bias (value of 1.0)
- Values raised to a power for each degree (x^1, x^2, x^3, etc…)
- Interactions between all pairs of features (x1*x2, x1*x3, etc…)

Polynomial Regression - Feature Interaction Example

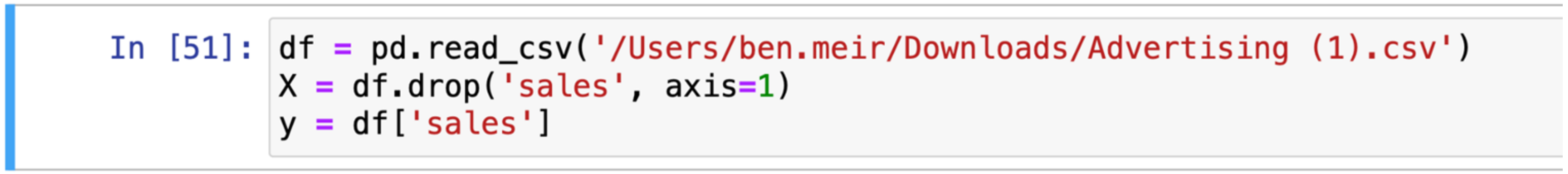
For example → Let's say we apply polynomial regression on data set containing 2 features (A & B).

The features that will be added to the model will be → 1 (bias), A, B, A^2, A*B, B^2.

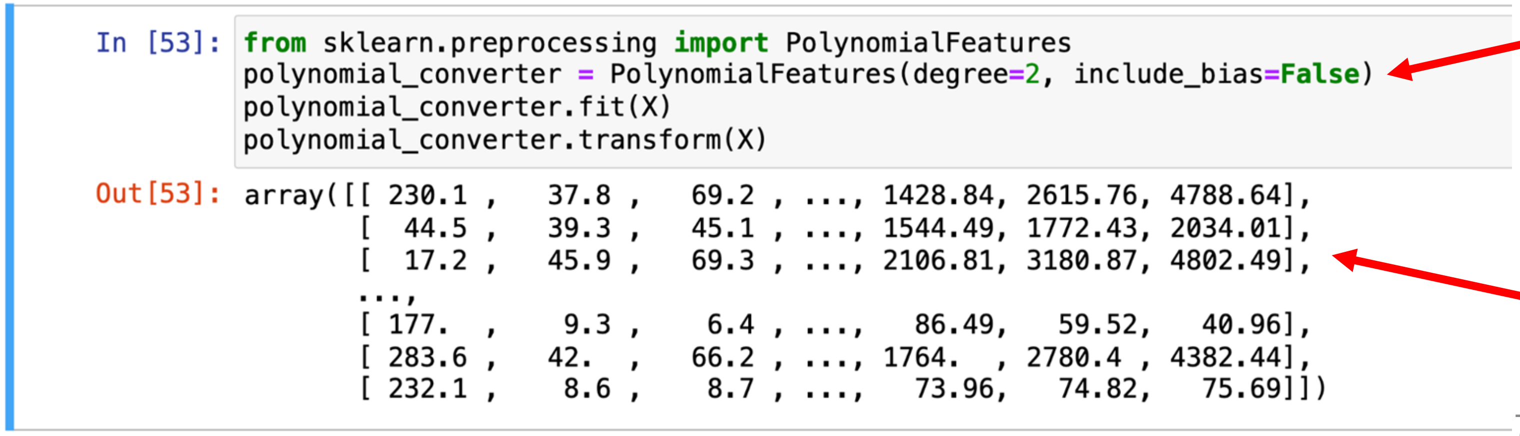
For each feature our model will generate a beta coefficient and the matched linear regression line.

Now let's take a real example and apply polynomial regression.

In this example we will use the 'Advertising.csv' file:



```
In [51]: df = pd.read_csv('/Users/ben.meir/Downloads/Advertising (1).csv')
         X = df.drop('sales', axis=1)
         y = df['sales']
```



```
In [53]: from sklearn.preprocessing import PolynomialFeatures
         polynomial_converter = PolynomialFeatures(degree=2, include_bias=False)
         polynomial_converter.fit(X)
         polynomial_converter.transform(X)

Out[53]: array([[ 230.1 ,   37.8 ,   69.2 , ..., 1428.84, 2615.76, 4788.64],
                [  44.5 ,   39.3 ,   45.1 , ..., 1544.49, 1772.43, 2034.01],
                [  17.2 ,   45.9 ,   69.3 , ..., 2106.81, 3180.87, 4802.49],
                ...,
                [ 177.  ,    9.3 ,    6.4 , ...,   86.49,   59.52,   40.96],
                [ 283.6 ,   42.  ,   66.2 , ..., 1764.  , 2780.4 , 4382.44],
                [ 232.1 ,    8.6 ,    8.7 , ...,   73.96,   74.82,   75.69]])
```

The fit() method determines how features need to be added.
The transform() method generate the numbers and apply them to the new dataframe

We managed to transform our original data set with 3 features to a new data set contain 9 features (each row has 9 values)

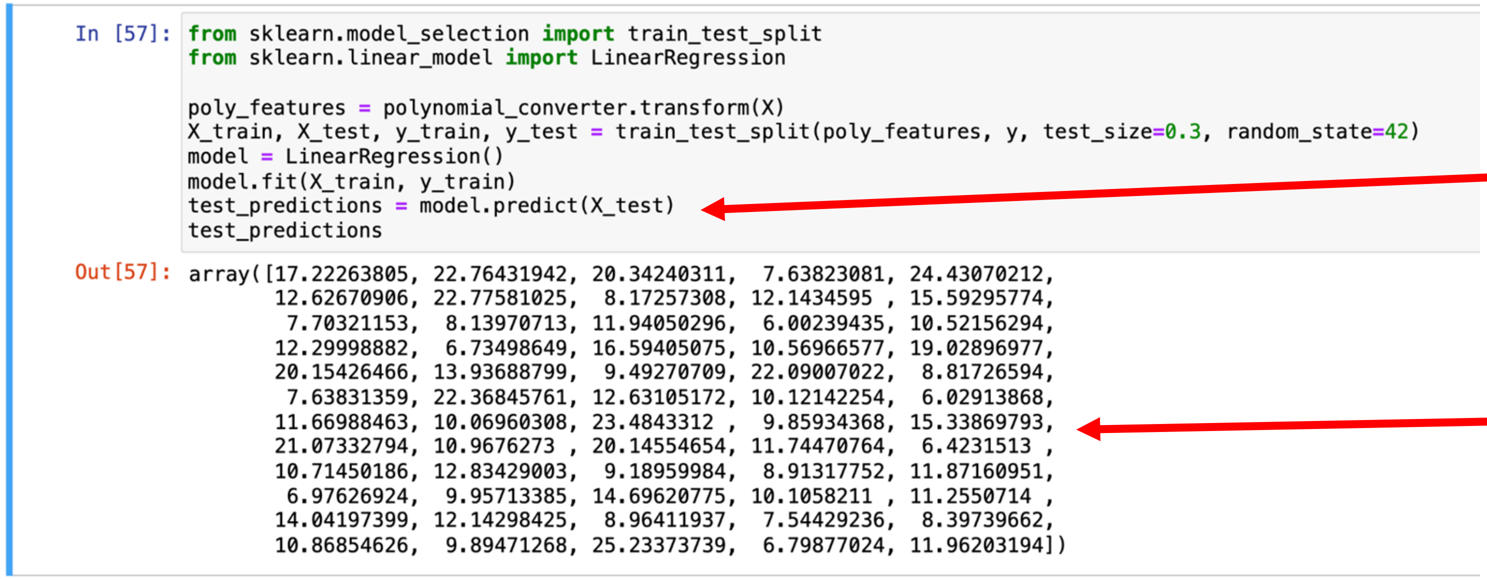
Polynomial Regression - Feature Interaction Example

Why we got 9 features instead the original 3 (A, B, C)?

The polynomial regression added the following features in this order:

A, B, C, A^2, A*B, A*C, B^2, B*C, C^2 → In total 9 different features.

Now that we have the transform data set we can apply linear regression modeling same way as before:



```
In [57]: from sklearn.model_selection import train_test_split
         from sklearn.linear_model import LinearRegression

         poly_features = polynomial_converter.transform(X)
         X_train, X_test, y_train, y_test = train_test_split(poly_features, y, test_size=0.3, random_state=42)
         model = LinearRegression()
         model.fit(X_train, y_train)
         test_predictions = model.predict(X_test)
         test_predictions

Out[57]: array([17.22263805, 22.76431942, 20.34240311,  7.63823081, 24.43070212,
                12.62670906, 22.77581025,  8.17257308, 12.1434595 , 15.59295774,
                 7.70321153,  8.13970713, 11.94050296,  6.00239435, 10.52156294,
                12.29998882,  6.73498649, 16.59405075, 10.56966577, 19.02896977,
                20.15426466, 13.93688799,  9.49270709, 22.09007022,  8.81726594,
                 7.63831359, 22.36845761, 12.63105172, 10.12142254,  6.02913868,
                11.66988463, 10.06960308, 23.4843312 ,  9.85934368, 15.33869793,
                21.07332794, 10.9676273 , 20.14554654, 11.74470764,  6.4231513 ,
                10.71450186, 12.83429003,  9.18959984,  8.91317752, 11.87160951,
                 6.97626924,  9.95713385, 14.69620775, 10.1058211 , 11.2550714 ,
                14.04197399, 12.14298425,  8.96411937,  7.54429236,  8.39739662,
                10.86854626,  9.89471268, 25.23373739,  6.79877024, 11.96203194])
```

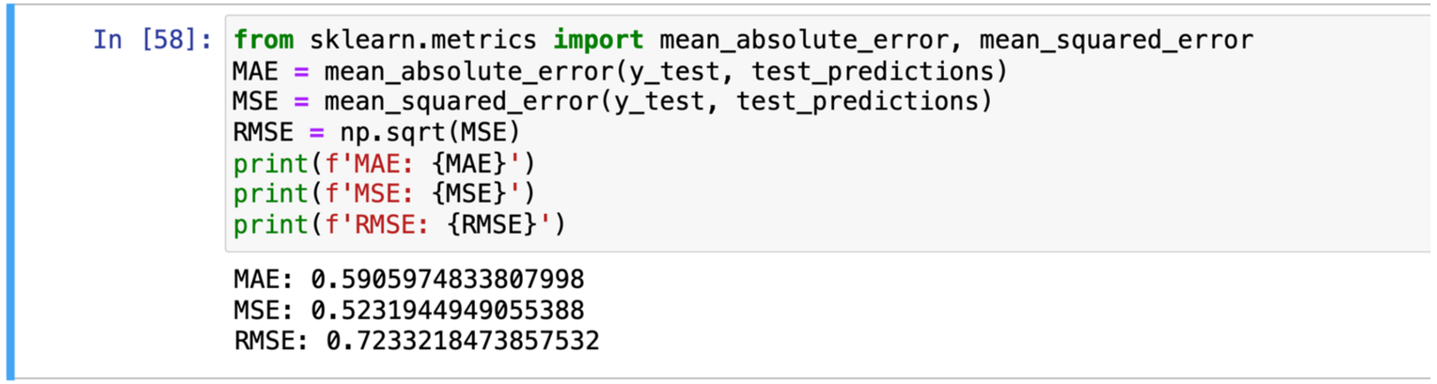
We apply train / test split and linear regression machine learning on the new data set.

The model predictions to the test set

Polynomial Regression - Feature Interaction Example

Once we have the model predictions we can evaluate the model performance using error metrics.

Let's get the model MAE, MSE, and RMSE values:



```
In [58]: from sklearn.metrics import mean_absolute_error, mean_squared_error
         MAE = mean_absolute_error(y_test, test_predictions)
         MSE = mean_squared_error(y_test, test_predictions)
         RMSE = np.sqrt(MSE)
         print(f'MAE: {MAE}')
         print(f'MSE: {MSE}')
         print(f'RMSE: {RMSE}')

         MAE: 0.5905974833807998
         MSE: 0.5231944949055388
         RMSE: 0.7233218473857532
```

We got that error metrics values of the polynomial regression are lower than the error metrics values of the linear regression we calculated earlier.

Previously MAE: 1.511

Previously RMSE: 1.948

New MAE: 0.5905

New RMSE: 0.7233

Note: Comparison is possible only if we choose the same parameters and the same data set.

Model Overfitting & Underfitting

Overfitting and **underfitting** are two common problems in supervised machine learning where the model learns too much or too little from the training data.

Overfitting → This occurs when the model learns the details and noise in the training data such that it negatively impacts the performance of the model on new, unseen data. The model becomes excessively complex and includes detailed patterns or noise that does not generalize well to new data. In other words, the model "fits" too well to the training data and performs poorly on the test data or in real-world application.

Underfitting → This happens when the model doesn't learn enough from the training data, failing to capture all relevant relations between inputs and outputs. The model is too simplified and therefore performs poorly on both the training and testing data. It lacks generalization from the training data and thus also performs poorly on unseen data.

Bias-Variance Tradeoff

When using polynomial regression we increase the model complexity. We now examine and train our model on higher degree features and also on correlations between features. This can lead to overfitting, the model might become "too good" at fitting the training data, including its noise and outliers, so in case we want to use polynomial regression we need to choose wisely the polynomial degree.

The bias-variance tradeoff → The bias-variance tradeoff is the point where we are adding just the right amount of complexity to our model which can be generalized well to unseen data.

Bias → refers to the error introduced by approximating a real-world problem, which may be extremely complicated, by a much simpler model. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting), leading to poor performance on both the training and test data.

Bias-Variance Tradeoff

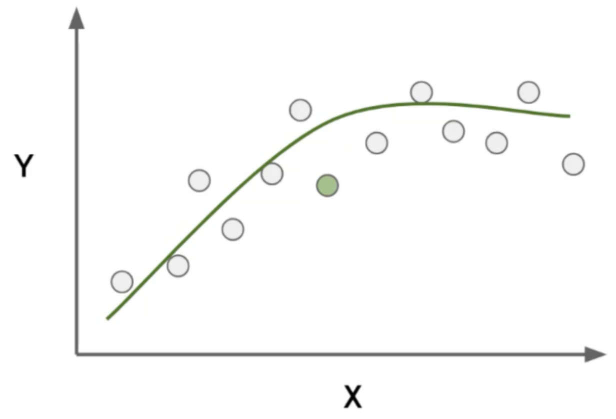
Variance → on the other hand, refers to the amount by which our model would change if we estimated it using a different training dataset. High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs (overfitting). This typically happens when the model is overly complex and performs very well on the training data but poorly on any unseen data.

In the context of the bias-variance tradeoff, <u>If we increase the complexity of the model further, variance will increase and bias will decrease leading to overfitting. Conversely, if we reduce the complexity of the model, bias will increase and variance will decrease leading to underfitting</u>.

In order to better understand overfitting and underfitting models we will visualize the linear regression line of each model using a line chart.

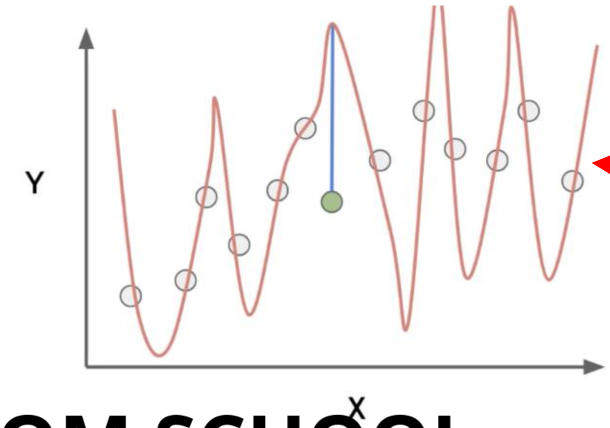
Model Overfitting & Underfitting - Visualization

Let's first take a look on a **good fitting model:**



We can see that the linear regression line is hitting close to all observation points and when a new unseen point added the model corresponding prediction fit well to it

Now Let's take a look on a **overfitting model:**

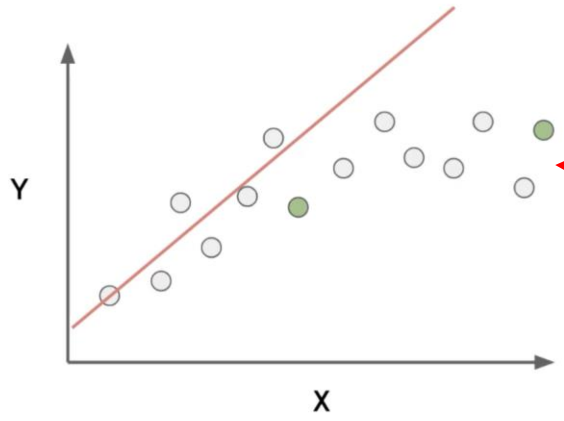


This model hit every single point so it's RMSE will be 0, But we can see that this regression line "fit" to good to the data it was trained on, resulting overfitting.
New unseen data won't behave this way and will result to a high error (as we can see in the green point)

ECOM SCHOOL
המכללה למקצועות הדיגיטל וההייטק

Model Overfitting & Underfitting - Visualization

Finally, let’s take a look on an **underfitting model:**



We can see that the linear regression line referees to the data linear correlation between the features and the label, resulting a simpler modeling on a more complex behavior.
This can lead to underfitting, the model is very simple and not examine additional complexities between the features.

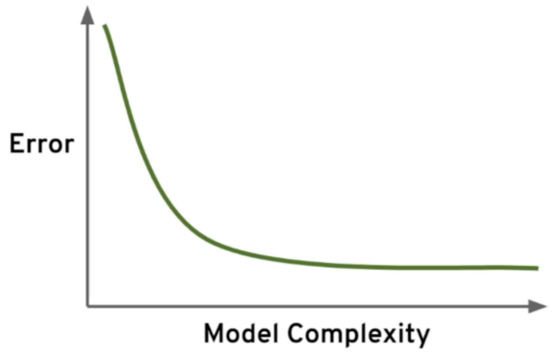
So how can we tell if our model is overfitting or underfitting?

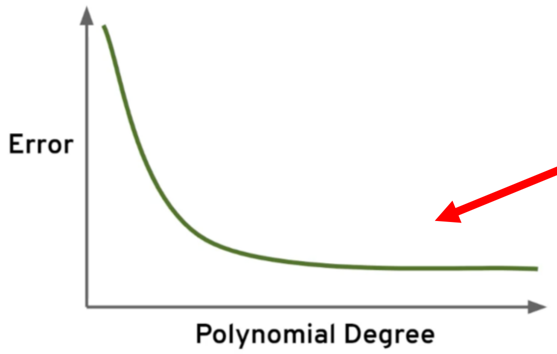
Discover underfitting → We can see underfitting when the model perform poorly on both the training and the testing data sets (easier to discover).

Discover overfitting → The model perform very well on the training and testing data sets but perform poorly on new unseen data (harder to discover).

Finding Overfitting & Underfitting

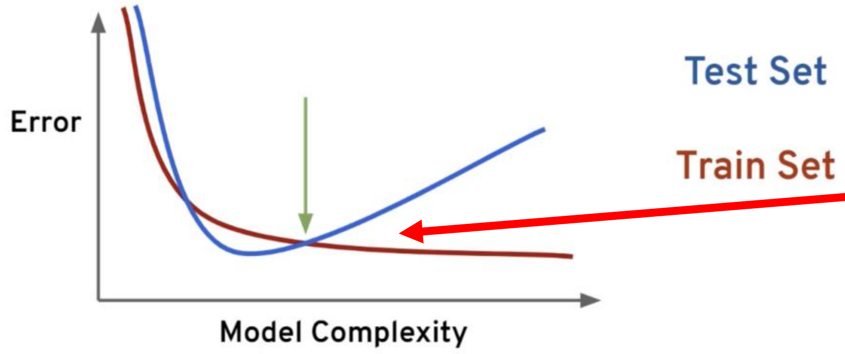
When dealing with high complexity models (models with multiple features and correlation between features) the way we can spot good modeling fit is if when we increase the model complexity, the model error going down accordingly.





In polynomial regression, increasing the model degree result increasing the model complexity

If we will see that we have a point which our error "spike" on, this point will be our maximum degree point



We can see that we have a point which adding more complexity to the model result spike in the error metric. meaning we don't want to pass this point

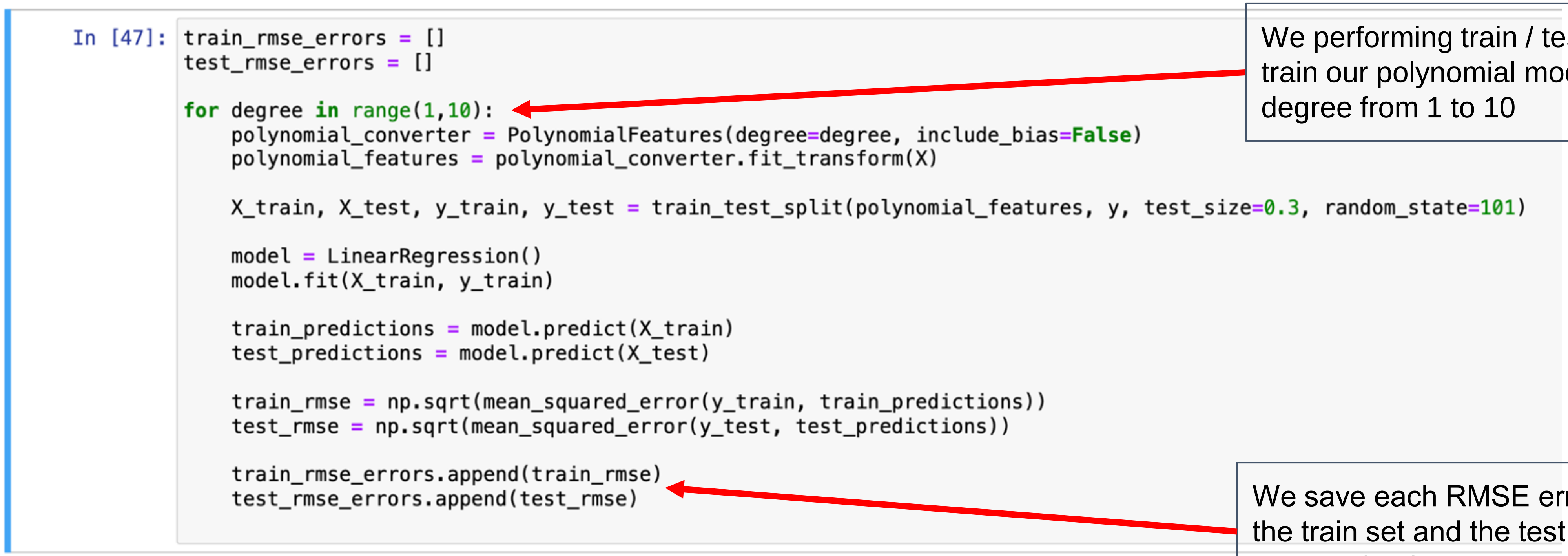
Find Most Fit Polynomial Degree

In order to find the correct polynomial degree which will add complexity to the model but not as much as it will become overfitting, we need to execute the following steps:

- Create different polynomial regression models with different degree
- Perform train / test split
- Fit all the polynomial models on the train set
- Store the RMSE for **both** the train set and and the test set for all the polynomial models
- Plot the results (error vs polynomial degree)
- See if there is a point which the RMSE spikes, you want to choose the degree before that spike

Find Most Fit Polynomial Degree - Example

Let’s continue with our campaign expenses example. In the previous example we decided on 2 degree polynomial modeling. Now what we want to do is to see what is the most ‘fit’ degree that we can chose to our model.



```
In [47]: train_rmse_errors = []
         test_rmse_errors = []

         for degree in range(1,10):
             polynomial_converter = PolynomialFeatures(degree=degree, include_bias=False)
             polynomial_features = polynomial_converter.fit_transform(X)

             X_train, X_test, y_train, y_test = train_test_split(polynomial_features, y, test_size=0.3, random_state=101)

             model = LinearRegression()
             model.fit(X_train, y_train)

             train_predictions = model.predict(X_train)
             test_predictions = model.predict(X_test)

             train_rmse = np.sqrt(mean_squared_error(y_train, train_predictions))
             test_rmse = np.sqrt(mean_squared_error(y_test, test_predictions))

             train_rmse_errors.append(train_rmse)
             test_rmse_errors.append(test_rmse)
```

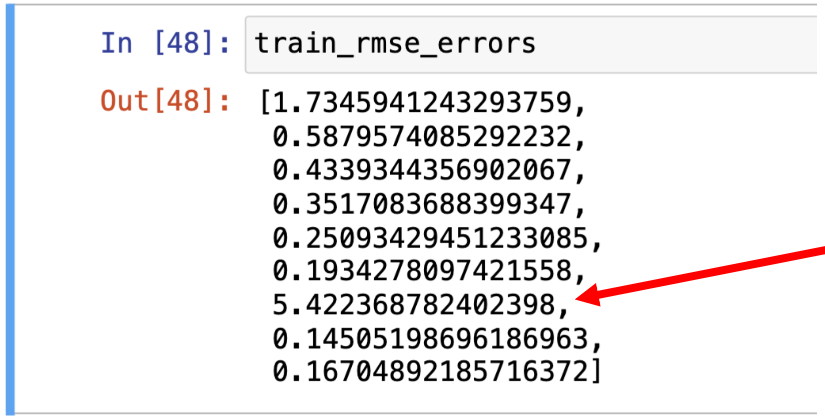
We performing train / test split and train our polynomial model on every degree from 1 to 10

We save each RMSE error on both the train set and the test set for each polynomial degree

ECOM SCHOOL
המכללה למקצועות הדיגיטל וההייטק

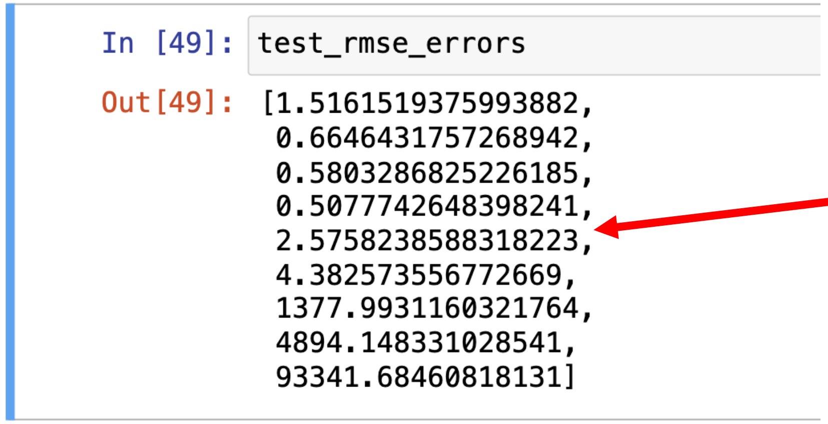
Find Most Fit Polynomial Degree - Example

What we have now is two separated arrays containing the RMSE of each polynomial degree:



```
In [48]: train_rmse_errors
Out[48]: [1.7345941243293759,
          0.5879574085292232,
          0.4339344356902067,
          0.3517083688399347,
          0.25093429451233085,
          0.1934278097421558,
          5.422368782402398,
          0.14505198696186963,
          0.16704892185716372]
```

We can see that on the 7th degree we got a large spike in our model RMSE so we don’t want to choose that degree or above it



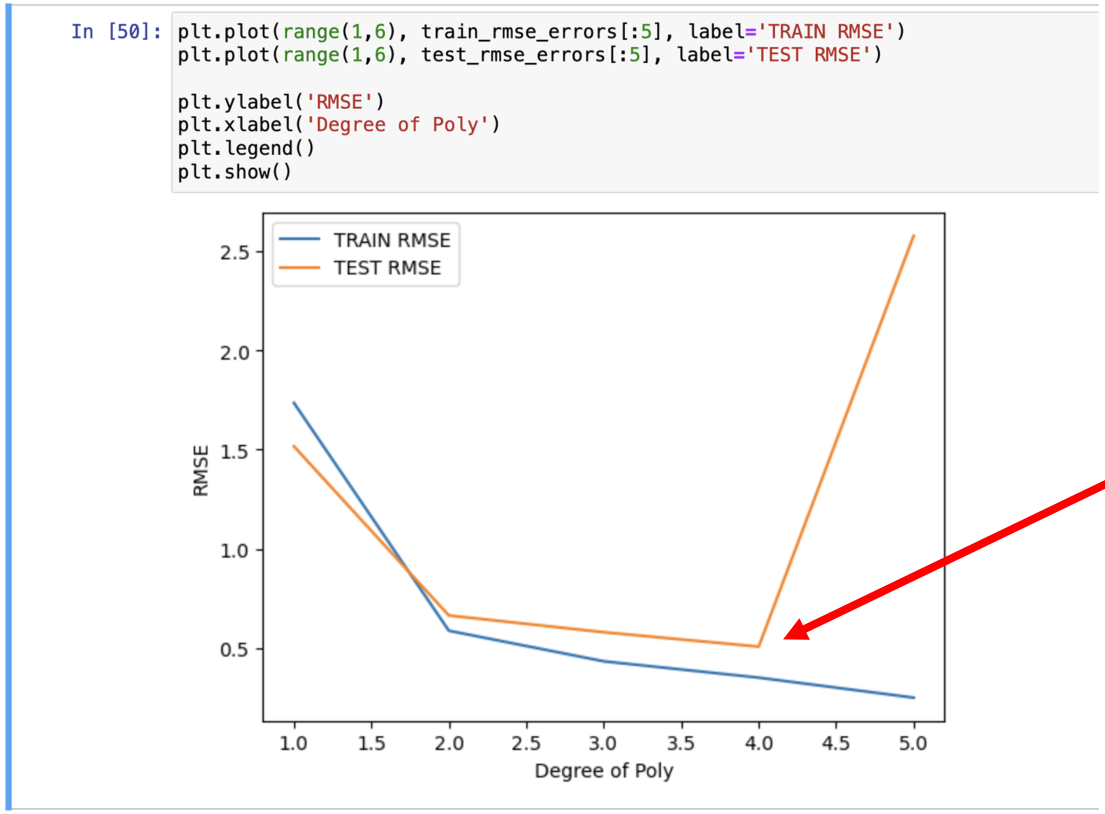
```
In [49]: test_rmse_errors
Out[49]: [1.5161519375993882,
          0.6646431757268942,
          0.5803286825226185,
          0.5077742648398241,
          2.5758238588318223,
          4.382573556772669,
          1377.9931160321764,
          4894.148331028541,
          93341.68460818131]
```

In the test RMSE array we got a spike on the 5th degree

ECOM SCHOOL
המכללה למקצועות הדיגיטל וההייטק

Find Most Fit Polynomial Degree - Example

Let’s plot the results we got so it will be easier for us to spot in which degree the spike in RMSE happen:



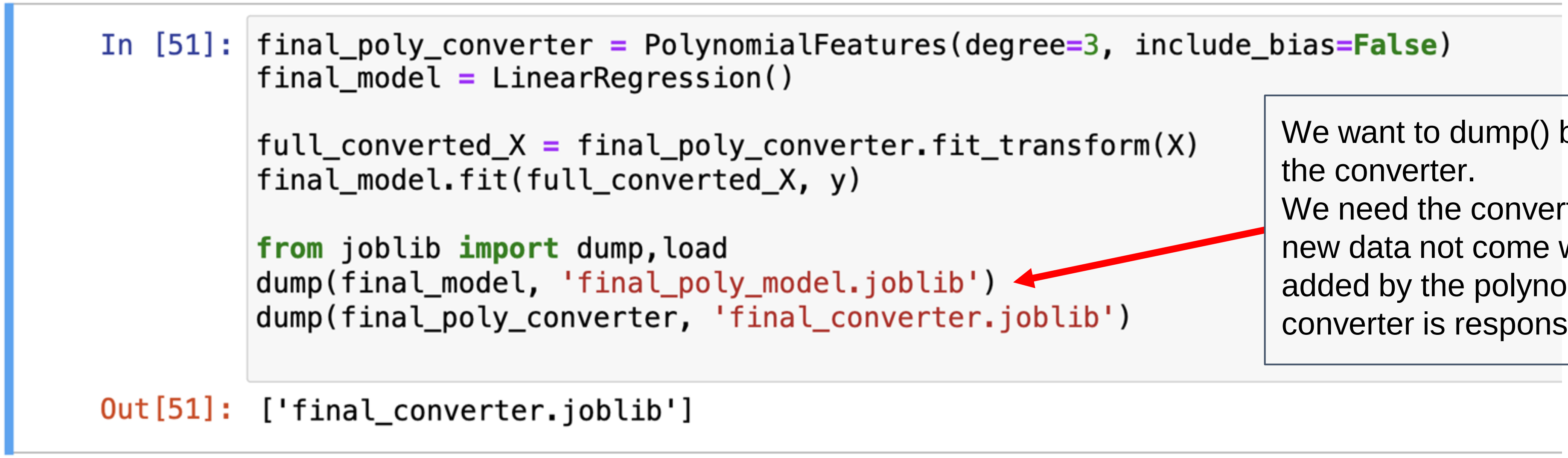
```
In [50]: plt.plot(range(1,6), train_rmse_errors[:5], label='TRAIN RMSE')
         plt.plot(range(1,6), test_rmse_errors[:5], label='TEST RMSE')

         plt.ylabel('RMSE')
         plt.xlabel('Degree of Poly')
         plt.legend()
         plt.show()
```

We can see that after the 4 degree the error in the test set exploded while the error in the train set kept going lower.

In conclusion → According to our investigation, we will want to choose polynomial degree of 2 or 3 so our model won’t be overfitting (above 3 degree) and won’t be underfitting (below 2 degree).

Find Most Fit Polynomial Degree - Example

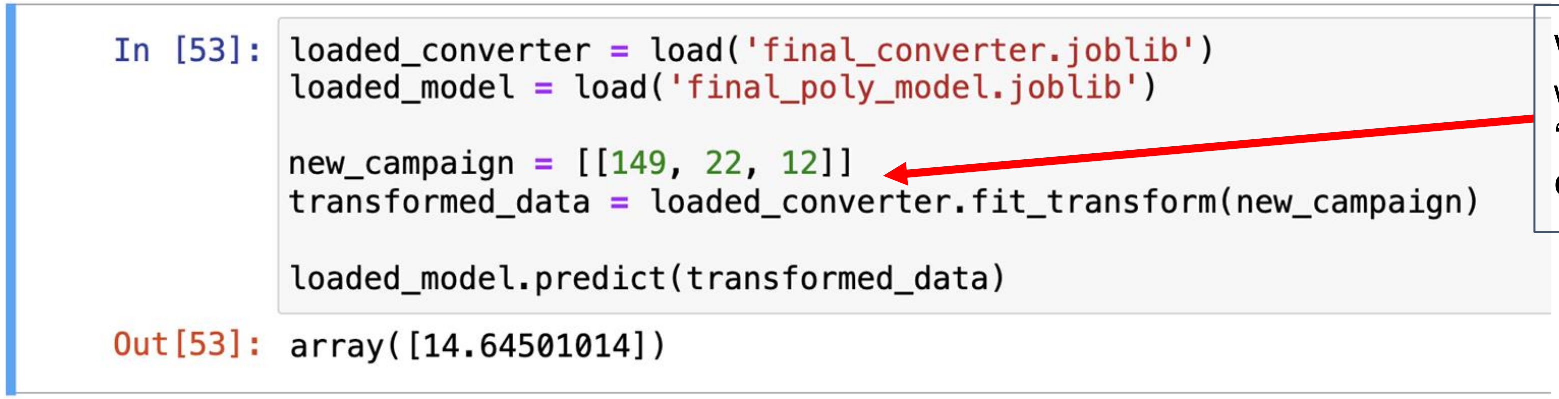
Now that we found our best fit degree, let's deploy our polynomial regression model:



```
In [51]: final_poly_converter = PolynomialFeatures(degree=3, include_bias=False)
         final_model = LinearRegression()

         full_converted_X = final_poly_converter.fit_transform(X)
         final_model.fit(full_converted_X, y)

         from joblib import dump,load
         dump(final_model, 'final_poly_model.joblib')
         dump(final_poly_converter, 'final_converter.joblib')

Out[51]: ['final_converter.joblib']
```

We want to dump() both our model and the converter.
We need the converter because the real new data not come with all the features added by the polynomial degree, the converter is responsible for that.

In order to load again the model and use it, we need to run the following code:



```
In [53]: loaded_converter = load('final_converter.joblib')
         loaded_model = load('final_poly_model.joblib')

         new_campaign = [[149, 22, 12]]
         transformed_data = loaded_converter.fit_transform(new_campaign)

         loaded_model.predict(transformed_data)

Out[53]: array([14.64501014])
```

When using the model with new data we first need to convert it using the 'final_converter' and then predict the converted data using the model itself

Class Exercise - Polynomial Regression

Instructions:

Use the data set provided in the previous class exercise and use **polynomial regression** model training.

Hours of calling customers: [2, 3, 4, 5, 6, 1.5, 5, 7, 8, 10]

Money earned: [50K$, 70K$, 90K$, 100K$, 110K$, 40K$, 110K$, 130K$, 145K$, 180K$]

- Train the model on 6 different polynomial regression degrees (from 1 to 6 included)
- Find what is the best fit degree for polynomial regression training
- Base your answer on plot visualizations
- Create a final model using the degree you chose and deploy it
- Import your final model from the joblib file and load it back to your working area
- Use the import model to predict how much money will be earned with the following new hours of calling customers [5.5, 15]

Class Exercise Solution - Polynomial Regression

ECOM SCHOOL

המכללה למקצועות הדיגיטל וההייטק