Comparing Linear Regression and Polynomial Regression in forecasting sales from media spending

I. Introduction

Businesses have been spending millions on different marketing channels, namely television, radio, social media, and influencers, to promote their products and service. Forecasting sales based on media spend with high accuracy has been a knotty problem for businesses. According to Korn Ferry, fewer than 25% of sales organizations is reported to predict sales with the accuracy of 75% or greater [1]. Machine learning can be leveraged to make the prediction process more efficient and reliable.

The machine learning problem will be illustrated further in section two – *problem formulation*. Section three will elaborate on the *methods* while *results and conclusion* will be explained in section four. Appendix and reference lists will also be included at the end of the report.

II. Problem Formulation

The main objective of this machine learning project is to predict sales based on media spend data and measure the accuracy of these predictions from two methods: Linear Regression and Polynomial Regression. The feature of data points used in these models includes the expenditure (numerical, in million) on television, social media, and radio promotion while the label is the number of sales (numerical, in million) gained from the allocated media budget.

III. Methods

To conduct the process of this project, Panda's package [2] was used for the preprocessing of the data; Matplotlib package [3] was imported for producing plots; Sci-kit learn package [4] was used for fitting, training, and testing the models.

3.1. Data Preprocessing

For the *data preprocessing stage,* dataset on media spends and sales can be obtained from Kaggle: "Dummy Marketing and Sales Data" [5] and there are 4572 datapoints with five columns labeled as 'TV', 'Social Media', 'Radio', 'Influencer' and 'Sales'. Firstly, for consistency in data, 'Influencer' column was removed as it contained non-numerical data. Also, null and NaN numbers were detected and removed. The data were then inspected once more to make sure there are no longer any NaN datapoints. The final and clean version of data contains 4546 rows and 4 columns.

3.2. Feature Selection

For *feature selection*, the relationship between the features and label should be reviewed. The correlation between each feature: 'TV', 'Social Media' and 'Radio' and label 'Sales' was visualized.

From the visualization, all the features show strong correlation to labels and both linear and curvilinear relationship are also detected.

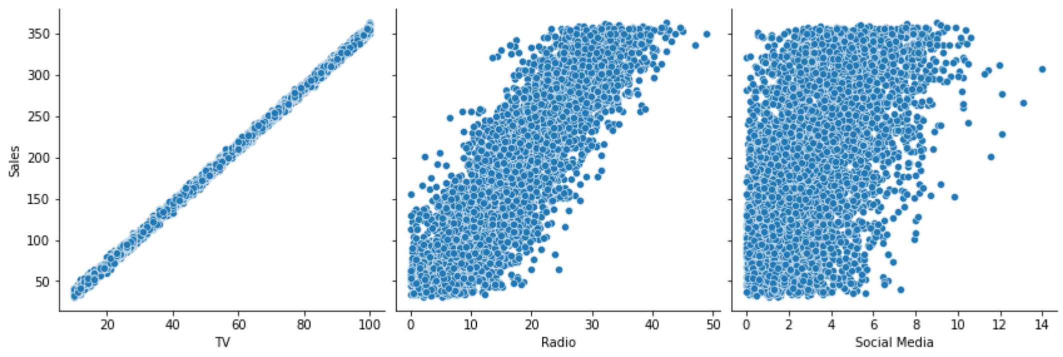


Fig.1. The relationship between the allocated budget of each medium and sales

There are three rationales for choosing Linear Regression and Polynomial Regression as my Machine Learning model. Firstly, these two models are chosen for how simple and interpretable they are. They can be used to estimate the relationships between dependent and independent variables. This is important in practice because businesses are interested in the underlying logic behind the workings of a model [6] and how sales are illustrated to be directly associated with their activities. Secondly, Linear Regression is likely to be the best model to illustrate the linear relationship observed from the visualization, while regarding the Polynomial Regression, the degree in the model can be increased, which in turn tends to increase the performance of the model [7].

3.3. Linear Regression model

The hypothesis space of a linear regression model is all functions of the form

$$\hat{y} = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n$$

where $\hat{y}$ is the predicted label, β_0 is a constant term, β_i are coefficients for different features, and x_i are the values of different features.

The loss function for the model is mean squared error (MSE)

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_{i)}^2$$

where y_i is the real label value and $\hat{y}_i$ is the predicted label value. It is used for regression fitting because it is normally used in Linear Regression [8] and also the default option in the Python package.

To perform the linear regression, the sklearn Python library and its LinearRegression() class was used.

3.4. Polynomial Regression model

The hypothesis space of Polynomial Regression model is constituted by polynomial maps

$$H^{(n)} = \left\{ h^{(w)}: \mathbb{R} \to \mathbb{R}: h^{(w)}(x) = \sum_{r=1}^{n} w_r \, x^{r-1} \text{, with some } w = (w_1, \dots, w_n)^T \in \mathbb{R}^n \right\}$$

Regarding loss function, because polynomial regression models are trained with different polynomial degrees, average squared error was used to measure the quality of the prediction [8]. It is calculated by summing all the datapoints, subtracting the predicted value and squaring that difference with the hypothesis

$$L_{MSE}\ (h) = \frac{1}{N} \sum_{i=1}^{N} (y_i - h(x_i))^2$$

For the process of model validation, the data were divided into three parts: training set (50%) and the remaining set (50%) with validation set (80%) and test set (20%). That specific ratio is chosen as it gives an adequate representative training set and when the remaining set is split into validation and test, the validation set remains large enough to provide meaningful validation results. Different orders of polynomials (d = 3, 5, 10) are fitted to the training set by minimizing the mean squared loss. K-fold Cross Validation was also applied: The entire dataset is divided evenly into k subsets - folds. The learning and validation of a hypothesis out of a given hypothesis space is then repeated k times. During each repetition, we use one-fold as the validation set and the remaining k−1 folds as a training set. The training and validation error is averaged through over all repetitions [8] From the validation errors calculated, the best Polynomial model is with degree of 3.

IV. Results and conclusion

The result of two machine learning models can be seen in the following table.

Linear Regression

Training error	Validation error	Test error
8.791817169141236	7.997390296381026	8.8411714150953

Polynomial Regression (with degree 3)

Training error	Validation error
8.64564	8.71924

To finalize on one machine learning models, validation errors of two given methods are compared as the validation set is used to estimate prediction error for model selection [9] With a smaller validation error, Linear Regression model is a better model with its test set comprising 20% of data points that have neither been used to train and its test error is 8.8411714150953.

There are many improvements that can be tried in the future. The problem can also be considered as a classification problem instead of a regression problem and the column "Influencer" can be included in the dataset through normalization. The training error was smaller than the validation error, which signals the problem of overfitting. Collecting more data can be considered for future revision. Another suggestion is to use a loss function that is more robust to outliers.

Reference

[1] kornferry.com. (n.d.). The top four challenges in sales forecasting. [online] Available at: https://www.kornferry.com/insights/featured-topics/sales-transformation/the-top-4-challenges-in-sales-forecasting [Accessed 30 Mar. 2022].

[2] kaggle.com. (n.d.). Dummy Marketing and Sales Data. [online] Available at: https://www.kaggle.com/harrimansaragih/dummy-advertising-and-sales-data [Accessed 07 Feb. 2022].

[3] https://pandas.pydata.org/pandas-docs/stable/index.html

[4] https://matplotlib.org/index.html

[5] https://scikit-learn.org/stable/index.html

[6] Shin, T. (2021). 3 Reasons Why You Should Use Linear Regression Models Instead of Neural Networks. [online] Medium. Available at: https://towardsdatascience.com/3-reasons-why-you-should-use-linear-regression-models-instead-of-neural-networks-16820319d644.

[7] Hershy, A. (2019). Simple Linear vs Polynomial Regression. [online] Medium. Available at: https://towardsdatascience.com/linear-vs-polynomial-regression-walk-through-83ca4f2363a3.

[8] A. Jung, *Machine Learning: The Basics*, http://mlbook.cs.aalto.fi

[9] Hastie, T., Tibshirani, R. and Friedman, J. (2009). The elements of statistical learning, second edition: data mining, inference, and prediction. New York: Springer.

Appendices - Coding Parts

March 30, 2022

[61]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, accuracy_score
from sklearn.linear_model import LinearRegression
import seaborn as sns
```

1 Importing dataset

[62]:
```
df = pd.read_csv('ad_budget.csv')
df.head(5)
```

[62]:
```
     TV      Radio  Social Media Influencer       Sales
0  16.0   6.566231      2.907983       Mega   54.732757
1  13.0   9.237765      2.409567       Mega   46.677897
2  41.0  15.886446      2.913410       Mega  150.177829
3  83.0  30.020028      6.922304       Mega  298.246340
4  15.0   8.437408      1.405998      Micro   56.594181
```

[63]:
```
df.shape
```

[63]:
```
(4572, 5)
```

2 Processing dataset

[64]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4572 entries, 0 to 4571
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   TV            4562 non-null   float64
 1   Radio         4568 non-null   float64
 2   Social Media  4566 non-null   float64
 3   Influencer    4572 non-null   object
 4   Sales         4566 non-null   float64
dtypes: float64(4), object(1)
memory usage: 178.7+ KB
```

[65]:
```
df = df.drop(['Influencer'], axis = 1)
```

[66]:
```
df = df.dropna()
```

[67]:
```
df.isnull()
```

[67]:
```
         TV  Radio  Social Media  Sales
0     False  False         False  False
1     False  False         False  False
2     False  False         False  False
3     False  False         False  False
4     False  False         False  False
…       …     …            …       …
4567  False  False         False  False
4568  False  False         False  False
4569  False  False         False  False
4570  False  False         False  False
4571  False  False         False  False

[4546 rows x 4 columns]
```

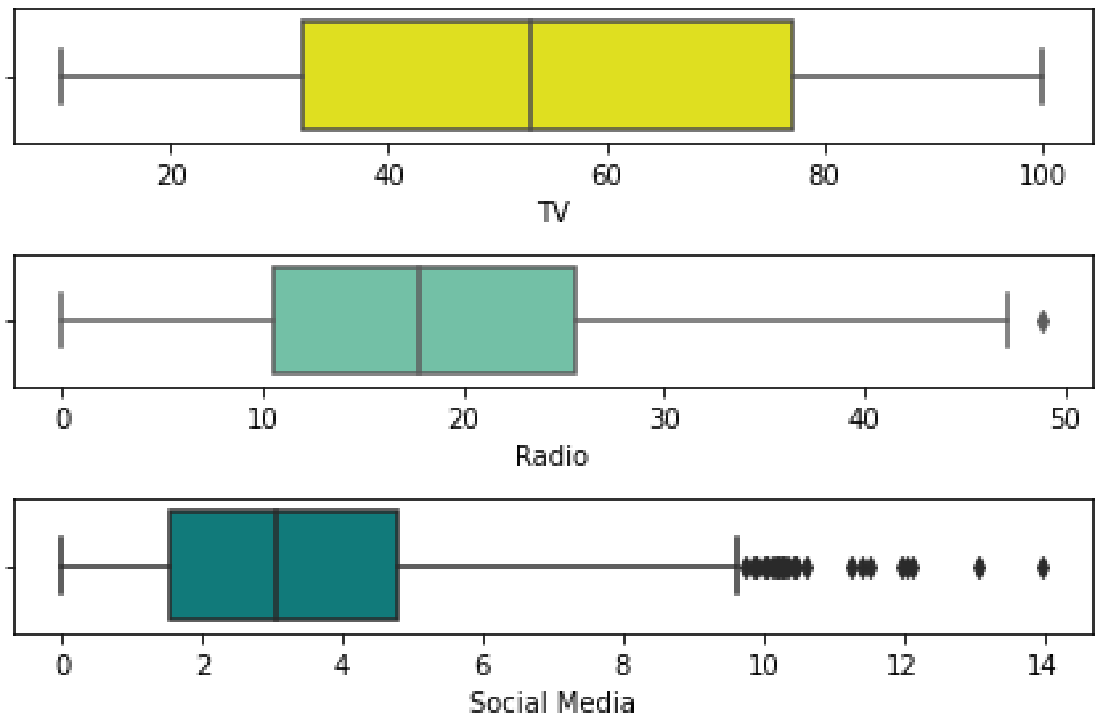
[68]:
```
df.describe()
```

[68]:
```
                TV        Radio  Social Media        Sales
count  4546.000000  4546.000000   4546.000000  4546.000000
mean     54.062912    18.157533      3.323473   192.413332
std      26.104942     9.663260      2.211254    93.019873
min      10.000000     0.000684      0.000031    31.199409
25%      32.000000    10.555355      1.530822   112.434612
50%      53.000000    17.859513      3.055565   188.963678
75%      77.000000    25.640603      4.804919   272.324236
max     100.000000    48.871161     13.981662   364.079751
```

3 Visualizing dataset

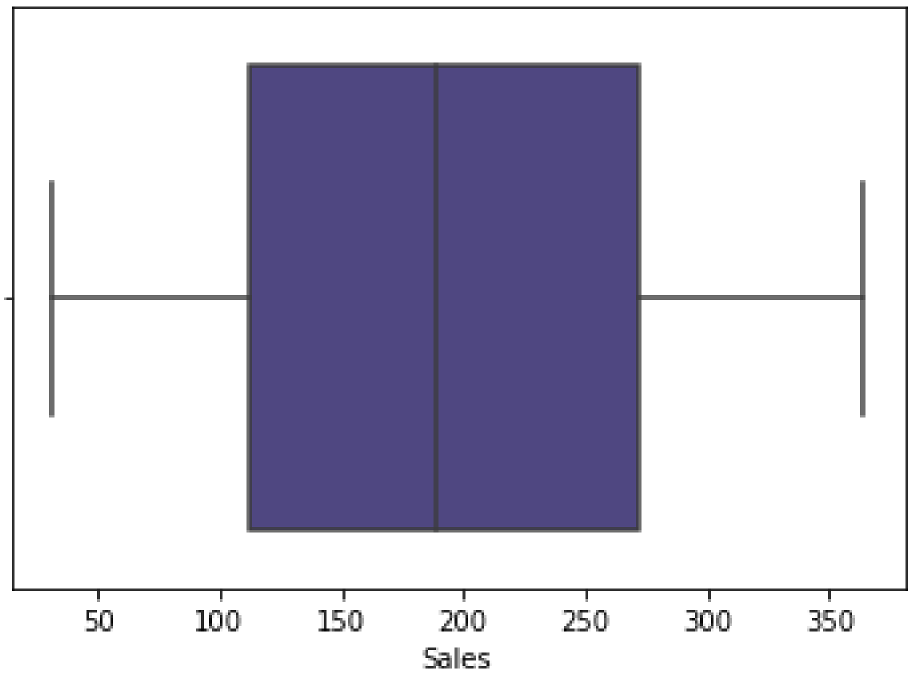
[69]:
```
fig, axs = plt.subplots(3, figsize =(6,4))
plt1 = sns.boxplot(df['TV'], ax = axs[0], color = 'yellow')
plt2 =sns.boxplot(df['Radio'], ax = axs[1], color = 'mediumaquamarine')
plt3 =sns.boxplot(df['Social Media'], ax = axs[2], color = 'darkcyan')
plt.tight_layout()
```

```
/opt/conda/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
/opt/conda/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
/opt/conda/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

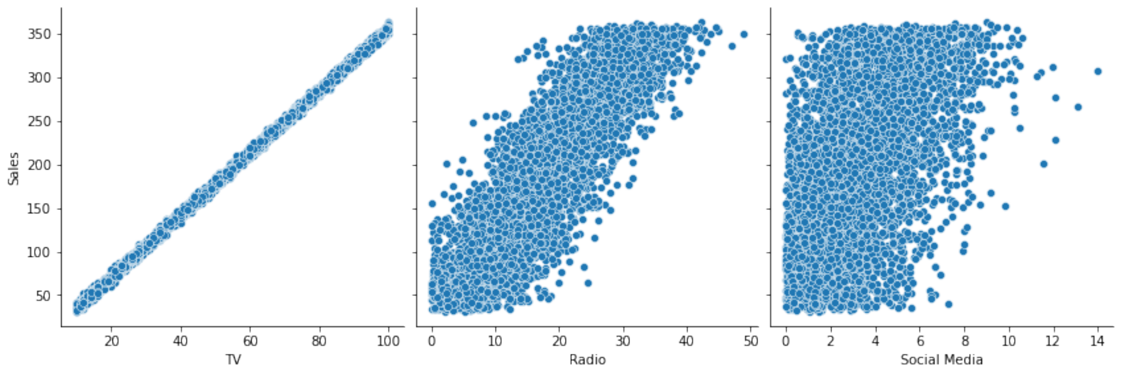


[70]:
```python
sns.boxplot(df['Sales'], color = 'darkslateblue')
plt.show()
```

```
/opt/conda/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



```
[71]: sns.pairplot(df,x_vars = ['TV', 'Radio', 'Social Media'], y_vars = 'Sales',␣
       ↪height = 4, aspect = 1, kind = 'scatter')
      plt.show()
```

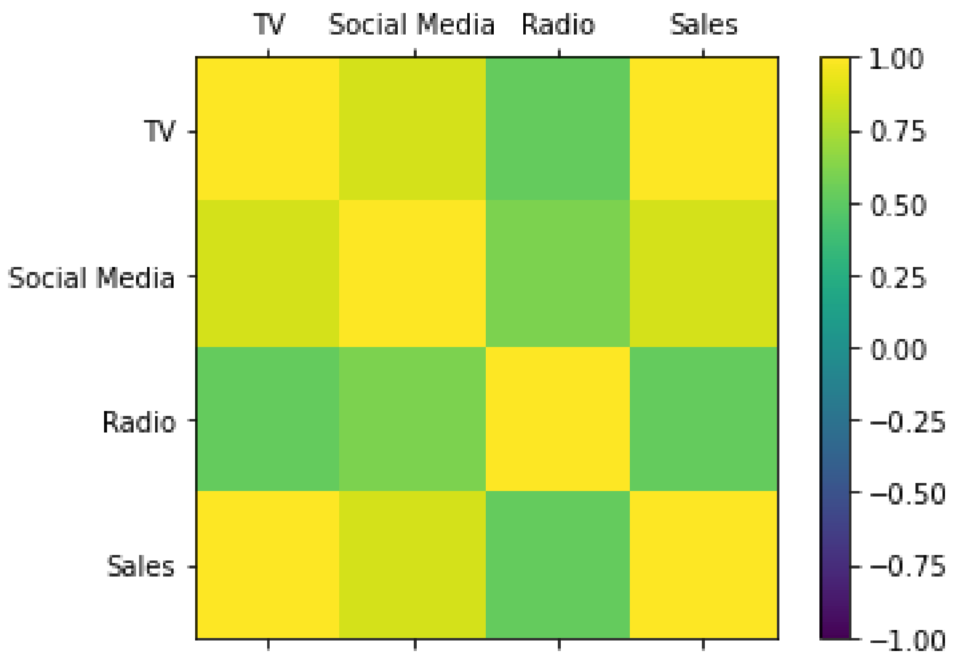


```
[72]: correlation = df.corr()
      names = ['TV', 'Social Media', 'Radio', 'Sales']

      fig = plt.figure()
      ax = fig.add_subplot(111)
      cax = ax.matshow(correlation, vmin=-1, vmax=1)
      fig.colorbar(cax)
```

```
ticks = np.arange(0,4,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(names)
ax.set_yticklabels(names)

plt.show()
```



4 Preparing Features and Labels

[73]:
```
X = df.drop(['Sales'], axis = 1).to_numpy()
y = df['Sales'].to_numpy()
```

[46]:
```
from sklearn.model_selection import train_test_split

X_train, X_rem, y_train, y_rem = train_test_split(
        X, y, test_size=0.5, random_state=42)

X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem, test_size=0.2,␣
 ↪random_state=42)
```

5 Training and testing Machine Learning models

5.1 Linear Regression

5.1.1 Training error

```
[47]: regr = LinearRegression()
      regr.fit(X_train, y_train)

      y_pred_train = regr.predict(X_train)

      tr_error = mean_squared_error(y_train, y_pred_train)


      print('The training error is ', tr_error)
```

```
The training error is  8.791817169141236
```

5.1.2 Validation error

```
[49]: y_pred_val = regr.predict(X_val)
      val_error = mean_squared_error(y_val, y_pred_val)

      print('The validation error is ', val_error)
```

```
The validation error is  7.997390296381026
```

5.1.3 Test error

```
[50]: y_pred_test = regr.predict(X_test)
      test_error = mean_squared_error(y_test, y_pred_test)

      print('The test error is ', test_error)
```

```
The test error is  8.8411714150953
```

5.2 Polynomial regression

First, polynomial regression models with different polynomial degrees are trained on training dataset. Then, by calculatimg validation errors from different techniques (K-Fold Cross Validation), the optimal poly degree is chosen.

5.2.1 Training and Validation errors

```
[51]: degrees = [3, 5, 10]

      tr_errors = []
      val_errors = []

```

```
for i, degree in enumerate(degrees):

    lin_regr = LinearRegression(fit_intercept=False)

    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.fit_transform(X_val)
    y_pred_val = lin_regr.predict(X_val_poly)
    val_error = mean_squared_error(y_val, y_pred_val)

    tr_errors.append(tr_error)
    val_errors.append(val_error)
```

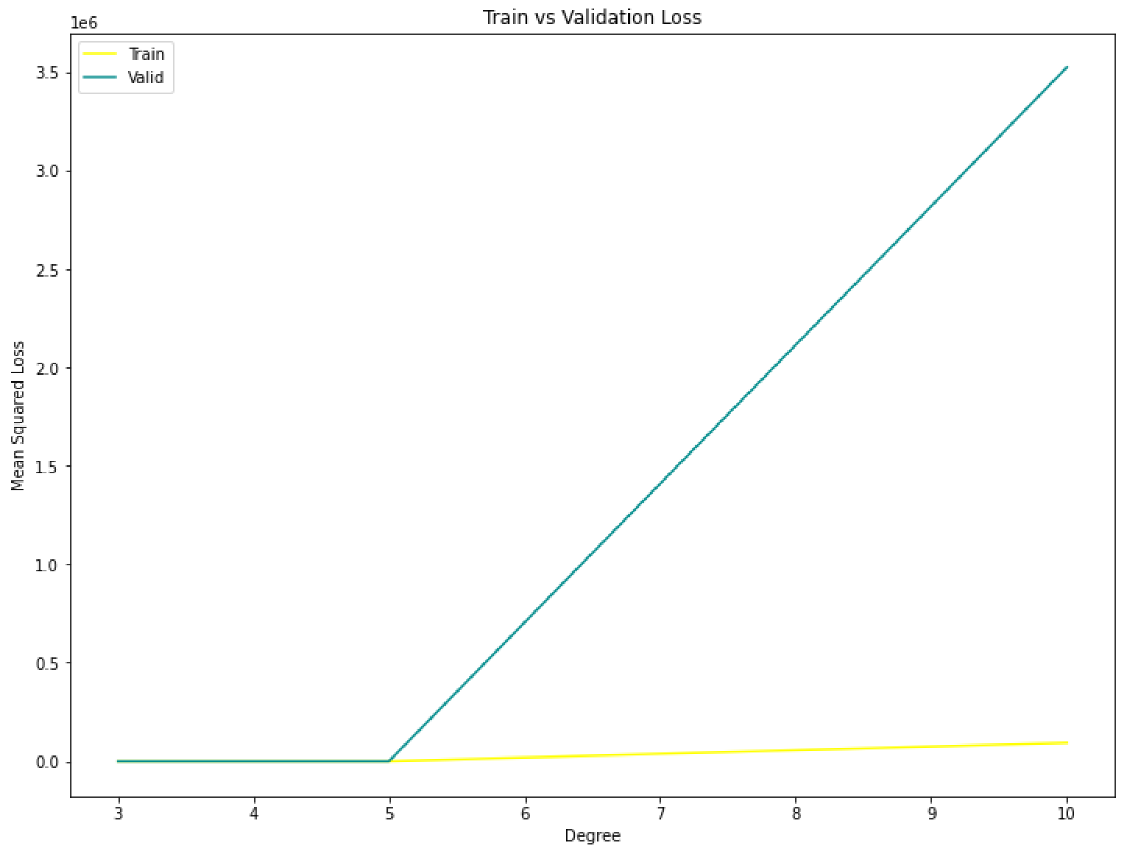
[52]:
```
print(tr_errors)
print(val_errors)
```

```
[8.739030803778675, 8.669393614418945, 93238.10194996397]
[8.081814697238608, 8.183914498100581, 3522318.07597522]
```

[53]:
```
plt.figure(figsize=(12, 9))

plt.plot(degrees, tr_errors, label = 'Train', color = 'yellow')
plt.plot(degrees, val_errors,label = 'Valid', color = 'darkcyan')
plt.legend(loc = 'upper left')

plt.xlabel('Degree')
plt.ylabel('Mean Squared Loss')
plt.title('Train vs Validation Loss')
plt.show()
```



[54]:
```
for i, degree in enumerate(degrees):
    print(f"For polynomial degree {degree}: the training error is␣
 ↪{tr_errors[i]}, the validation error is {val_errors[i]}")
```

```
For polynomial degree 3: the training error is 8.739030803778675, the validation
error is 8.081814697238608
For polynomial degree 5: the training error is 8.669393614418945, the validation
error is 8.183914498100581
For polynomial degree 10: the training error is 93238.10194996397, the
validation error is 3522318.07597522
```

5.2.2 Cross Validation

[55]:
```
from sklearn.model_selection import KFold
```

[56]:
```
k, shuffle, seed = 3, True, 42
kfold = KFold(n_splits=k, shuffle=shuffle, random_state=seed)
```

[57]:
```
degrees = [3, 5, 10]
```

```
tr_errors = {}
val_errors = {}


for i, degree in enumerate(degrees):
    tr_errors[degree] = []
    val_errors[degree] = []

    for j, (train_indices, val_indices) in enumerate(kfold.split(X)):


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

        lin_regr = LinearRegression(fit_intercept=False)
        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)
```

[58]:
```
tr_errors, val_errors
```

[58]:
```
({3: [8.613377386059836, 8.581110405451424, 8.742419197040144],
  5: [8.544458968191316, 8.48620143447923, 8.625146172979852],
  10: [70854.8551985008, 161286.32931056467, 72438.61252427335]},
 {3: [8.804462961630454, 8.845089894122674, 8.50816078756804],
  5: [8.92169602885427, 8.97759601843698, 8.758916525779474],
  10: [1560101.0450739798, 11218784.365404453, 4025501.9284319207]})
```

[59]:
```
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}, avg train error = {average_train_error[degree]:.
 ↪5f}, "
          f"avg val error = {average_val_error[degree]:.5f}")
```

```
Degree 3, avg train error = 8.64564, avg val error = 8.71924
Degree 5, avg train error = 8.55194, avg val error = 8.88607
Degree 10, avg train error = 101526.59901, avg val error = 5601462.44630
```

5.2.3 Final test error

From the validation errors calculated above, it can be seen that the best Polynomial model is with degree of 3. The test error is calculated based on this model.

[60]:
```
lin_regr = LinearRegression(fit_intercept=False)

poly = PolynomialFeatures(degree=3)
X_train_poly = poly.fit_transform(X_train)
lin_regr.fit(X_train_poly, y_train)

X_test_poly = poly.fit_transform(X_test)
y_pred_test = lin_regr.predict(X_test_poly)
test_error = mean_squared_error(y_test, y_pred_test)

print("The test error is: ",test_error)
```

```
The test error is:  8.708604749280212
```