CST4050 - Individual Assessment (comp 2)

Student:

- Name: SARIMA

1. Loading data and preliminary analysis

The block below loads our data and seperates features (X) from the target attribute (y)

In [142…
```
import pandas as pd

data = pd.read_csv("C:/Users/sarim/Downloads/SCHOOL/MACHINE_LEARNING/COURSEWORK 2/data

X = data.drop("y", axis=1)  # select all columns except target ("y")
y = data[["y"]] # select only the target column "y"
```

The block below examines the target attribute "y"

In [143…
```
y.head()
```

Out[143]:

	y
0	32.175299
1	6.521285
2	5.688139
3	8.488786
4	63.570984

From the result above, we see that the target variable 'y' is numerical and continuous, therefore, the supervised problem at hand is a regression task. The objective is to predict a continuous variable.

To understand the frequency distribution of the outcome, the code below shows some statistics of the target variable

In [144…
```
y.describe()
```

Out[144]:

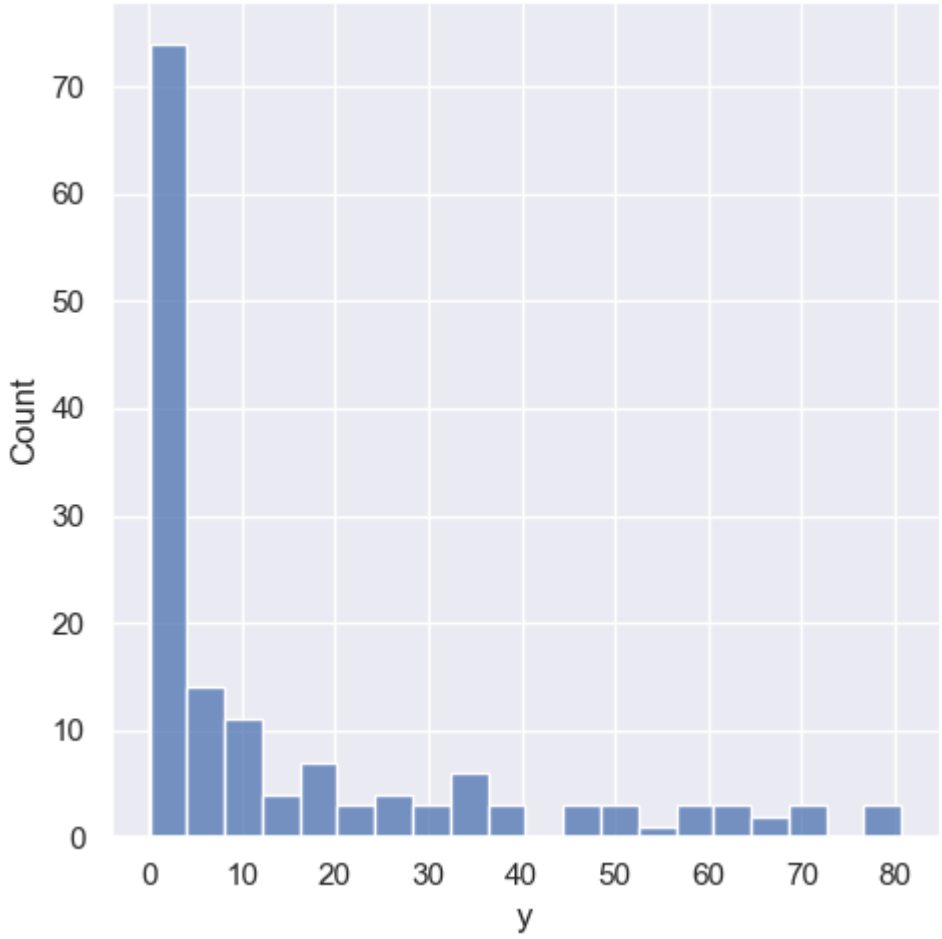
	y
count	150.000000
mean	15.573634
std	21.532456
min	0.081166
25%	0.855930
50%	4.188566
75%	22.455500
max	80.673096

Based on the summary statistics provided, the mean value, 15.573634 and standard deviation value of 21.532456 are not enough to determine the shape of the distribution or how the values are distributed within the range of 'y'. To get a detailed understanding of the frequency distribution of our target variable, we would need to visualize the distribution using a histogram.

In [145…

```
#load library for data visualization
import seaborn as sb
import matplotlib.pyplot as plt

hist = sb.displot(data=data, x='y', bins=20)
plt.show(hist)
```



Based on the histogram, the data for the target variable 'y' is left skewed, which means that some extremely large values in the range of 'y' are contributing to a high value of the mean and also that the variability in the data is a lot as indicated by the high standard deviation.

1.2. Checking the features

The code below inspects the feature data

In [146…
```
# Show the first five rows in X

X.head()
```

Out[146]:

	x1	x2	x3	x4	x5	x6	x7	x8
0	0.592109	0.545496	0.199054	2.370339	1.032510	1.137605	1.199802	-0.833456
1	0.911316	1.260952	0.446375	1.564526	0.820080	2.172268	1.441165	-0.373705
2	0.968683	3.744257	1.931173	1.472553	0.102181	4.712941	1.954805	1.828992
3	0.748656	2.741351	1.790573	1.696788	0.464028	3.490008	0.948294	1.326545
4	0.320502	3.196858	1.050494	2.813036	0.204284	3.517361	1.273237	0.846210

In [147…
```
#Show the number of rows and columns in X
```

```
X.shape
```

Out[147]:
```
(150, 8)
```

From the result, the X dataframe contains 150 observations(rows) and 8 features(columns).

In [148…
```
#show the 5-number summary of X

X.describe()
```

Out[148]:

	x1	x2	x3	x4	x5	x6	x7	x8
count	150.000000	150.000000	150.000000	150.000000	150.000000	150.000000	150.000000	150.000000
mean	0.526873	2.532959	1.004755	1.437147	0.608091	3.059833	1.005223	0.396664
std	0.287178	1.366323	0.591567	0.869658	0.365670	1.405730	0.418172	0.666874
min	0.000789	0.039299	0.022259	0.034008	0.000504	0.324586	0.097891	-1.012279
25%	0.286739	1.467144	0.484289	0.685930	0.263076	1.988572	0.748112	-0.100801
50%	0.538521	2.489126	1.025886	1.410161	0.628974	2.985679	0.979791	0.389450
75%	0.776970	3.650754	1.504459	2.163303	0.920787	4.241689	1.292235	0.892501
max	0.986514	4.981110	1.976849	2.993408	1.293620	5.960314	1.954805	1.910219

Based on the description above, it can be observed that the mean and standard deviation of the columns are not 0 and 1 respectively. This indicates that the data is not standardized since each column is on a different scale.

2. Training and testing a machine learning pipeline

Pipeline

The proposed pipeline is defined as follows:

1. StandardScaler: To standardize the features, addressing the left-skewed distribution and scaling the data to 0 and 1 for the mean and standard deviation respectively in order to have a more interpretable model.

2. LinearRegression: To fit a linear regression model on the transformed features.

Split the data into training and testing sets

In [149…
```
# load the neccesary library
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.3, random_state
```

The line of code above splits the dataset into 70% train set for training and 30% test set for testing the accuracy of the model

The code below computes the R^2 value, which we would use as our accuracy estimate for the proposed pipeline

In [150…

```
#Load relevant library

from sklearn.pipeline import Pipeline # Library to create a pipeline
from sklearn.preprocessing import StandardScaler  # Library to standardise features
from sklearn.linear_model import LinearRegression  # Library to perform linear regress
import warnings
warnings.filterwarnings('ignore')


# Creating the Pipeline
pipeline = Pipeline([('scaler', StandardScaler()), # scaling features
                     ('lm', LinearRegression())]) # train a linear regression model or

# fit the pipeline on the training data
pipeline.fit(X_train, y_train)


# Make predictions on train and test data
y_train_pred = pipeline.predict(X_train)
y_test_pred = pipeline.predict(X_test)

# Computing the accuracy
R2_train = pipeline.score(X_train, y_train)
R2_test = pipeline.score(X_test, y_test)

print("Train R-squared:", R2_train)
print("Test R-squared:", R2_test)
```

```
Train R-squared: 0.7684684691103149
Test R-squared: 0.7209460099934151
```

The train and test R-squared values are high and close, indicating that the model is neither overfitting nor underfitting. This suggests that the model has a good bias-variance tradeoff and a good fit.

However, the simplicity of the pipeline consisting only of Standardization and Linear Regression implies that the model may have a high bias but low variance, as linear models only recognize linear relationships between dependent(output) and independent(input) variables. Adding polynomial features can help to reduce bias and improve complexity by incorporating the modeling of non-linear relationships

3. Tuning the proposed machine learning pipeline

Since the feature space has a low dimension with a column size way less than number of rows, we would improve the complexity of the model by creating a new pipeline that includes polynomial regression.

This pipeline would first tune the hyperparameters(in this case the polynomial features degree) using K-Fold Cross Validation on the tuning set to find the best hyperparameters and finally validate the final model on the unseen validation set

This approach would improve's the model's complexity, its ability to capture non-linearity between the features and the target variable and help mitigate the chance of overfitting by analyzing the model's perfomance on hyperparameters on multiple folds of data and then give a better estimate of the model's performance based on the best hyperparameter.

Split the data into tuning and validation sets

For a better estimate of the accuracy on the model and to ensure our tuning and validation sets are representative of a good portion of the dataset, we would use 70% tuning set from our data to tune the hyperparameters using k-fold cross validation and use 30% validation set from our data to validate the final model.

In [151…
```
# Split data into 70% tuning, 30% validation
X_tuning, X_val, y_tuning, y_val = train_test_split(X, y, test_size=0.3, random_state=
```

In [152…
```
#load the neccessary library
from sklearn.model_selection import KFold
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score
import numpy as np


# Define a range of degrees to try
degrees = range(1, 3)

# Initialize lists to store scores for each degree
scores = []

# Use nested KFold cross-validation to evaluate the model for each degree of the polyr
kf = KFold(n_splits = 10, shuffle=True, random_state=10)

for degree in degrees:
    pipeline = Pipeline([
        ('poly', PolynomialFeatures(degree=degree)),
        ('scale', StandardScaler()),
        ('linreg', LinearRegression())
    ])
    degree_scores = []

    for train_idx, val_idx in kf.split(X_tuning):
        X_train_kf, y_train_kf = X_tuning.iloc[train_idx], y_tuning.iloc[train_idx]
        X_val_kf, y_val_kf = X_tuning.iloc[val_idx], y_tuning.iloc[val_idx]

        pipeline.fit(X_train_kf, y_train_kf)  # fit the pipeline

        y_val_kf_pred = pipeline.predict(X_val_kf) # make predictions

        score = r2_score(y_val_kf, y_val_kf_pred) # compute accuracy on test data

        degree_scores.append(score) #update degree_scores array with accuracy for each
```

```
    mean_degree_score = np.mean(degree_scores)
    scores.append(mean_degree_score) # update vector that stores the R2 scores(accurac
                                     # ...each degree with the best fit model score

# Find the degree with the best R2 score
best_degree = degrees[np.argmax(scores)]

print("Best Polynomial Features Degree: ", best_degree)
```

```
Best Polynomial Features Degree:  2
```

Based on the tuning results, it appears that the best hyperparameter value offering the best estimate of bias-variance trade off is a polynomial transformation to a degree of 2

Validation on unseen data

Below we evaluate the final model's performance on the unseen validation set using the best hyperparameter from the tuning phase

In [153…

```
# Build final model for evaluation
final_pipeline = Pipeline([
                                ('scale', StandardScaler()), # Standardize the data
                                ('poly', PolynomialFeatures(2)), # polynomial transfor
                                ('lr', LinearRegression()) # Fit a linear regression m
                               ])

# fit the pipeline
final_pipeline.fit(X_tuning, y_tuning)

# Make predictions
y_train_pred = final_pipeline.predict(X_tuning)
y_val_pred = final_pipeline.predict(X_val)

# Compute the accuracy
R2_score_train = final_pipeline.score(X_tuning, y_tuning)
R2_score_val = final_pipeline.score(X_val, y_val)



print("R2 on tuning data:", R2_score_train)
print("R2 on validation data:",  R2_score_val )
```

```
R2 on tuning data: 0.987707493326728
R2 on validation data: 0.9717922146279739
```

The R^2 values of the our first model are significantly lower than the R^2 values of the tuned model, confirming that the first model was underfitting the data.

In summary, the final tuned model is performing better than the non-tuned model with a better bias-variance tradeoff.

4. Model interpretation

To ensure complete model interpretability we would analyze and verify whether the conditions of regression models are met.

Below we first create a dataframe with the actual values, predicted values and residuals on our best model, which uses the validation data

In [154…
```
#Build data frame with actual and predicted target (validation) data.
y_val_pred = np.squeeze(y_val_pred)

verification = pd.DataFrame({"actual": y_val.y.values})
verification["fitted"] = y_val_pred
verification["residuals"] = verification["actual"] - verification["fitted"]
verification.describe()
```

Out[154]:

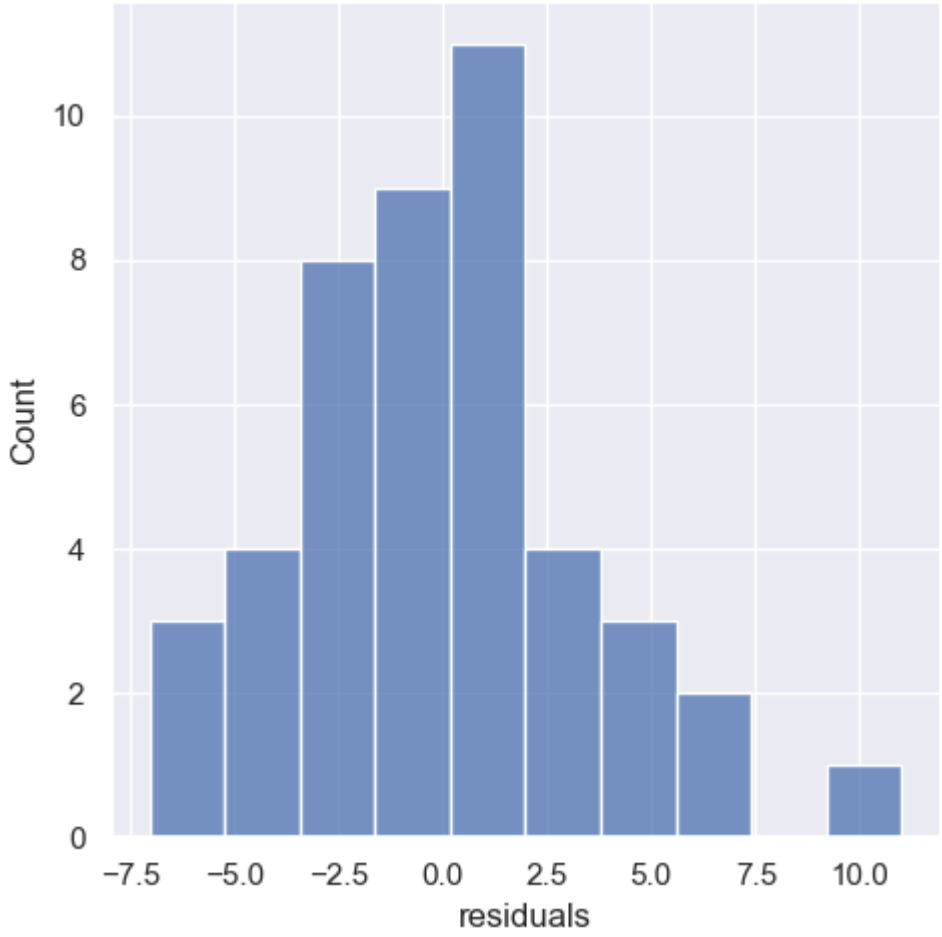
	actual	fitted	residuals
count	45.000000	45.000000	45.000000
mean	14.000149	14.015159	-0.015010
std	20.805416	20.013459	3.494273
min	0.081166	-4.466859	-7.033030
25%	0.843835	1.158141	-2.289989
50%	4.680310	5.292907	-0.197075
75%	14.089273	16.294860	1.717480
max	80.673096	69.677672	10.995424

The first condition is to check the normality of Residuals

Let's visualize the residuals below

In [155…
```
sb.set(rc={'figure.figsize':(4,3)})
sb.displot(data = verification, x="residuals", bins = 10)
```

Out[155]:
```
<seaborn.axisgrid.FacetGrid at 0x1a73fd416f0>
```

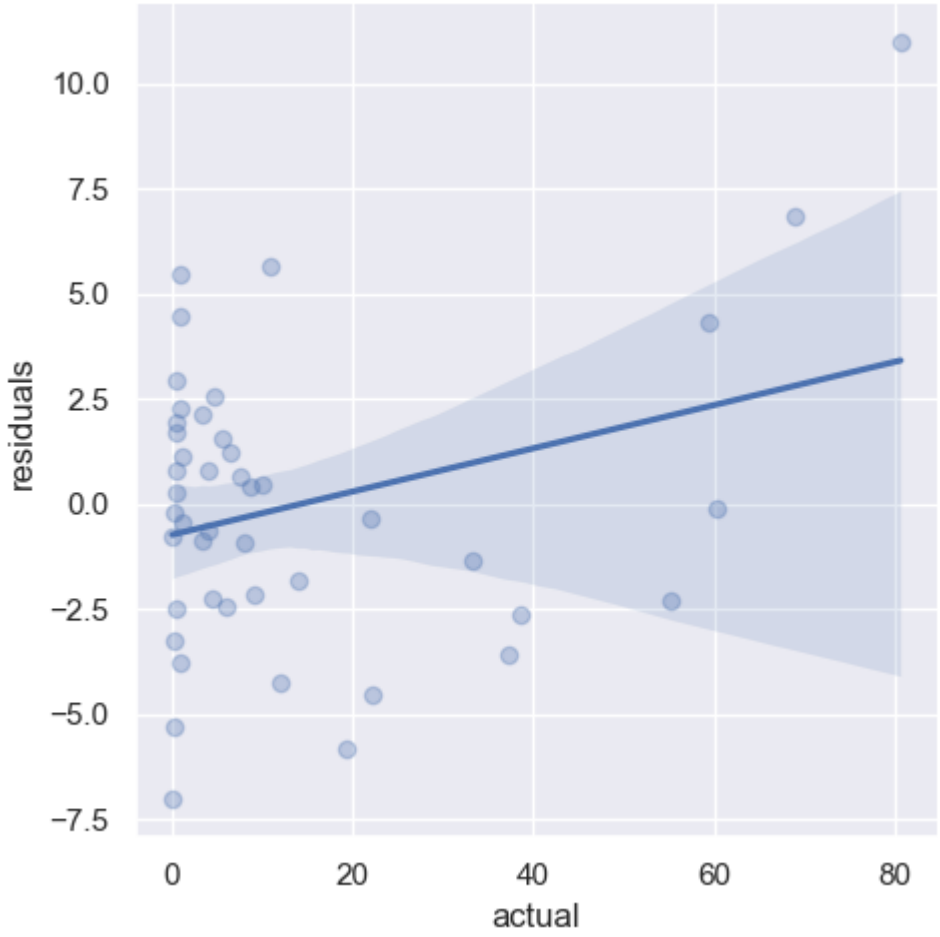


While the distribution may appear bell-shaped, the presence of an outlier indicate that the data is partially normally distributed. So therefore, the normality of residuals condition is slightly met.

Next we check for independence of Residuals(Homoscedasticity) by plotting a scattered plot to explore the relationship of residuals between actual and predicted values.

In [156…
```
sb.set(rc={'figure.figsize':(4,3)})
sb.lmplot(data=verification, x="actual", y="residuals", scatter_kws={'alpha':0.3})
```

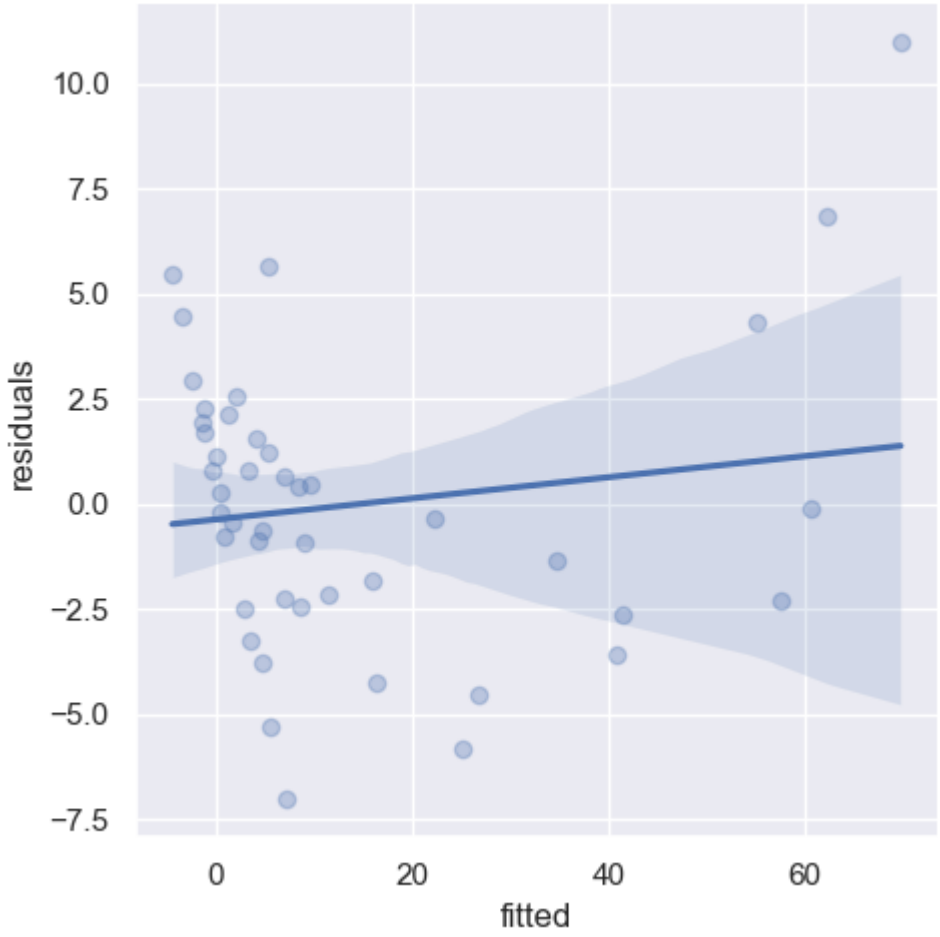
Out[156]:
```
<seaborn.axisgrid.FacetGrid at 0x1a73fb6f070>
```



Here we can see that the Residuals are not correlated with actual values and are completely random.

In [157…
```
sb.set(rc={'figure.figsize':(4,3)})
sb.lmplot(data=verification, x="fitted", y="residuals", scatter_kws={'alpha':0.3})
```

Out[157]:
```
<seaborn.axisgrid.FacetGrid at 0x1a73fe574c0>
```



We can also see the same for the predicted values, however, despite this randomness, there are some little signals indicating that there is room for improvement in our model

Finally, to verify if our model is interpretable, we would check to see if there is no multicolinearity among the predictors

In [158…
```
X_tuning.corr()
```

Out[158]:

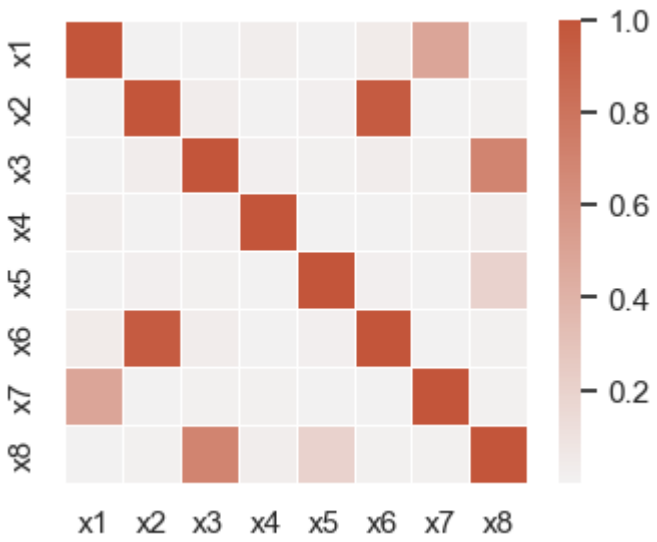
	x1	x2	x3	x4	x5	x6	x7	x8
x1	1.000000	0.006487	0.036280	-0.161786	0.006617	0.208506	0.698328	0.028974
x2	0.006487	1.000000	0.192409	-0.044470	0.128695	0.979353	-0.059266	0.101672
x3	0.036280	0.192409	1.000000	-0.136270	0.118482	0.195518	-0.122963	0.834137
x4	-0.161786	-0.044470	-0.136270	1.000000	0.066331	-0.076200	-0.088775	-0.159481
x5	0.006617	0.128695	0.118482	0.066331	1.000000	0.127207	-0.024002	-0.448841
x6	0.208506	0.979353	0.195518	-0.076200	0.127207	1.000000	0.083210	0.105297
x7	0.698328	-0.059266	-0.122963	-0.088775	-0.024002	0.083210	1.000000	-0.097329
x8	0.028974	0.101672	0.834137	-0.159481	-0.448841	0.105297	-0.097329	1.000000

Some features show strong correlations, Let's visualize the correlation matrix below

In [159…
```
# Generate a custom diverging colormap
cmap = sb.diverging_palette(230, 20, as_cmap=True)

sb.heatmap(X_tuning.corr()**2, cmap=cmap, center=0, square=True, linewidths=.5)
```

Out[159]:
```
<Axes: >
```



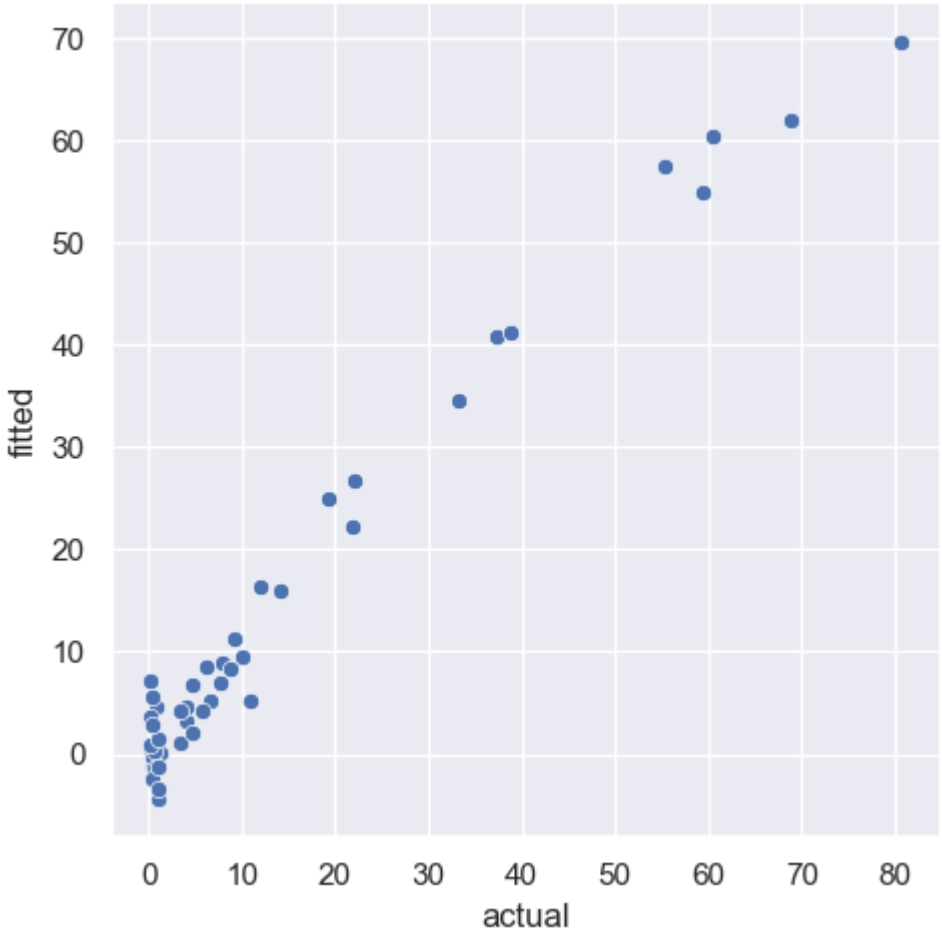
Based on the visualization, we can confirm that there is a significant amount of multicolinearity among the predictors.

In summary our proposed pipeline fits quite well on unseen data with an R^2 value of **0.917**, meaning that the model would be good for making predictions, however, the model does not meet all the conditions of a linear regression model, as there is multicolinearity amongst features indicating that the beta coefficients are unstable. This means that the model is not interpretable but has a good fit for making predictions.

we can visualize this in the scattered plot below to compare the actual and predicted target data

In [160…
```
sb.relplot(data=verification, x="actual", y="fitted")
```

Out[160]:
```
<seaborn.axisgrid.FacetGrid at 0x1a73fd41fc0>
```



From the plot above we can see that there is a good alignment between actual target and predicted target data, this shows that our model would have a good estimate of accuracy when it comes to predictions.

Now to analyze the inferred function, we'll evaluate the βs below

In [161…

```
# Get the coefficients
coef = final_pipeline.named_steps['lr'].coef_

# Print the coefficients
print(coef)
```

```
[[ 5.43511508e+09 -1.53732912e+08 -7.43750010e+08 -7.28124105e+08
   1.72272924e+01  4.49414388e+08  7.60448138e+08 -7.34128906e-02
   8.09071169e+08 -5.14285946e+11 -4.97675532e+12  3.96752265e+11
  -3.42387075e+08 -2.45175891e+11  5.08856137e+12 -1.98171860e+08
  -4.40549401e+11 -1.20400399e+13  1.91559609e+12 -1.65644681e+09
  -1.18375911e+12  2.46210543e+13 -9.58742802e+08 -2.12705404e+12
  -6.43350057e+12  4.07782628e+08  7.94113607e+12 -1.95848825e+12
  -6.08632680e+08  1.42967474e+13  1.23525870e+01 -2.51692505e+08
   1.69363614e+09 -1.49529608e-01 -4.53116668e+08 -2.45051823e+12
   1.21026477e+12  3.75661623e+08 -8.82353352e+12 -1.25870912e+13
   9.80267790e+08  2.17468087e+12  7.00010209e-01  6.76295634e+08
  -7.94268156e+12]]
```

Based on the coefficients, we can observe the polynomial transformation on our feature space from a low dimensional to a high dimensional space.

From the coefficients of the model we can infer the polynomial function below

$f(x) = \sum_{m=1}^{n} \beta_m \cdot x^m$, where β_n are the coefficients of the model and x is the features.

$$y = 5.43511508e+09 - 1.53732912e+08x1 - 7.43750010e+08x2 - 7.28124105e+08x3 + 08x6 - 7.34128906e-02x7 + 8.09071169e+08x8 - 5.14285946e+11x1^2 - 4.97675532 - 2.45175891e+11x1x5 + 5.08856137e+12x1x6 - 1.98171860e+08x1x7 - 4.40549401e$$

we can interprete some of the β coefficients as follows

- The β coefficient, +8.09071169e+08, of the x8 attribute is positive and large indicating that, x8 has a strong weight on predicting the outcome variable.
- Attributes with negative β coefficients shows that the feature has a negative effect on the outcome variable.

4. Discussion: pros/cons and time complexity of the designed pipeline

The final pipeline is human interpretable as it involved standardizing features using StandardScaler to ensure all features are on the same scale, it then transforms them into polynomial features of degree 2, which was selected based on the best R2 score obtained from cross-validation tuning, all to improve complexity, capture non-linear relationships as well to improve model performance, finally it fits a linear regression model on the transformed data.

Time complexity of the proposed pipeline

The time complexity for training a linear regression model with polynomial features is $O(n * m^d)$, where

- n is the number of observations
- m is the number of features, and
- d is the degree of the polynomial.

For the proposed pipeline, the highest time complexity comes from the nested For loop, which runs 10-fold cross-validation for each degree of polynomial features to be evaluated. The number of observations and the number of features increased to the degree of polynomial features (m^d) have the greatest influence on time execution.

The nested For Loop's high-level algorithm is as follows:

```
For each degree to evaluate:
    For each fold k:
        Split the data into train and test sets
            Standardize the data
            Transform the data using polynomial features
            Train a linear regression model
```

```
            Test the linear regression model
        Compute the mean accuracy across all folds for the current
degree
Choose the degree with the highest mean accuracy
```

Therefore, the time complexity for training the proposed pipeline having k-fold cross validation with polynomial features and linear regression is $O(k * d * n * m^d)$, where

- k is the number of folds
- d is the number of degrees to tune
- m is the number of features, and
- n is the number of observations.

Strengths and Limitations

Strengths:

- The pipeline perform k-fold cross validation to tune the hyperparameters that gives a better estimate of bias-variance tradeoff which helps to prevent underfitting or overfitting.
- The use of Polynomial Features allows for data complexity, better model performance and capturing the non-linear relationships between outcome and predictors.
- Data standardization would make sure that all the features are on the same scale, curbing the impact of features with high readings(outliers). This is also important as linear regression assumes all features have equal importance in predicting the target variable.
- The pipeline is a helpful tool for quick modelling because it is reasonably simple to set up and apply to fresh datasets.

Limitations:

- The model cannot be used for hypothesis testing due to the unstable coefficients(multicollinearity) β
- Polynomial regression is prone to overfitting on training data, specifically with higher degrees of polynomial features.
- The pipeline may not perform well if some interactions between features that are not captured by the polynomial features.
- The pipeline assumes that linear models are a suitable fit for the data, however this may not be the case for all datasets.
- The pipeline could be computationally expensive, particularly when working with large datasets and high degrees of polynomial features due to the time complexity.

Possible ways to improve the pipeline

- Implement regularization techniques, such as lasso regression, to reduce overfitting after the polynomial transformation step. This regularization approach can also serve as a feature selection technique to lower the dimensionality of the feature space.

- Applying other feature selection techniques such as Principal Component Analysis (PCA) and Variance Inflation Factor (VIF) before the polynomial transformation to address the issue of multicollinearity.