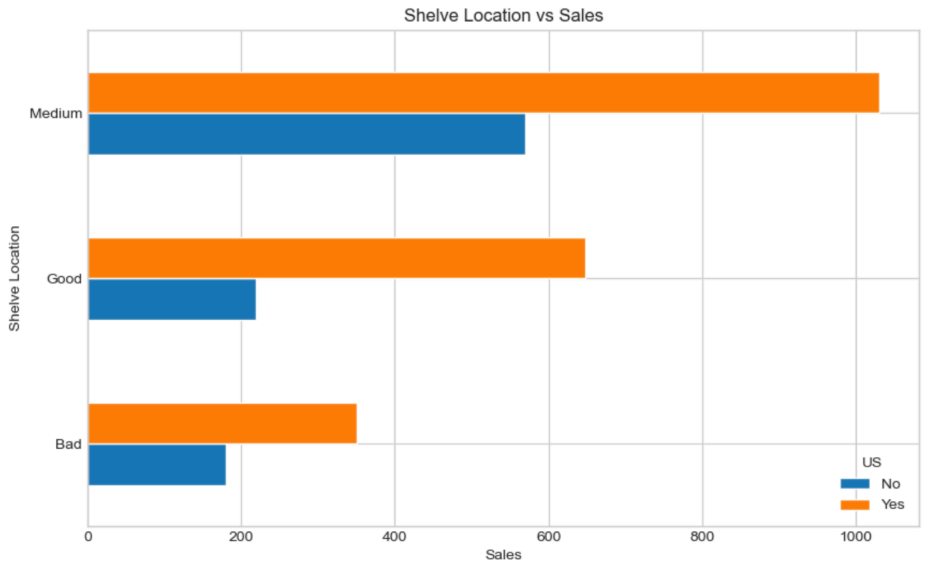
# Machine Learning
# Homework 3

1. ***EDA: Develop a python program that: [15 points]***
   a. ***Read the dataset and plot the group horizontal bar plot that shows the 'ShelveLoc' versus Sales with respect to the 'US' (legend). Which Shelve location has the highest sales and whether the best sales are inside the US or outside? Hint: You need to aggregate using summation method.***

⇨ **Code**

```
#==========================================================================================================
# Question 1
#==========================================================================================================

# 1. a
url='https://raw.githubusercontent.com/rjafari979/Information-Visualization-Data-Analytics-Dataset-/main/Carseats.csv'
df=pd.read_csv(url)
plt.style.use("seaborn-whitegrid")
sns.set_style("whitegrid")
group_data=df.groupby(['ShelveLoc','US'])['Sales'].sum().unstack()
group_data.plot(kind='barh', stacked=False, figsize=(10, 6))
plt.xlabel('Sales')
plt.ylabel('Shelve Location')
plt.title('Shelve Location vs Sales')
plt.show()
plt.grid(True)
max_sales_location = group_data.sum(axis=1).idxmax()
max_sales_value = group_data.loc[max_sales_location]
print(f"Shelve location with the highest sales  '{max_sales_location}'.")
print(max_sales_value)
```

⇨ **Output**



```
Shelve location with the highest sales  'Medium'.
US
No      569.52
Yes    1030.62
Name: Medium, dtype: float64
```

***b. Using the one-hot-encoding converts the qualitative features to quantitative features and makes it ready for regression analysis (zeros or ones). Display the first 5 rows of the converted features on the console.***

⇨ **Code**

```
# 1. b
df_one_hot_encode=pd.get_dummies(df,columns=['ShelveLoc','Urban','US'],drop_first=True)
converted_features = df_one_hot_encode[df_one_hot_encode.columns.difference(df.columns)]
print(converted_features.head())
```

⇨ **Output**

```
   ShelveLoc_Good  ShelveLoc_Medium  US_Yes  Urban_Yes
0               0                 0       1          1
1               1                 0       1          1
2               0                 1       1          1
3               0                 1       1          1
4               0                 0       0          1
```

***c. Standardize the dataset in the previous step and split the dataset into train-test 80-20. Turn the shuffle ON and use the random_state = 5805. Display the first 5 rows of the train and test set. Hint: The encoded features should not be standardized. Use the following for the split.***

## ⇨ Code

```
# 1. c
non_encoded_columns=['Sales','CompPrice','Income','Advertising','Population','Price','Age','Education']
scaler=StandardScaler()
df_standard=df_one_hot_encode;
df_standard[non_encoded_columns]=scaler.fit_transform(df_one_hot_encode[non_encoded_columns])
df_train,df_test=train_test_split(
    *arrays: df_standard,test_size=0.20,shuffle=True,random_state=5805
)
print(df_train.head())
print(df_test.head())
```

## ⇨ Output

```
        Sales  CompPrice    Income  ...  ShelveLoc_Medium  Urban_Yes  US_Yes
15   0.430292   1.568690  0.942452  ...                 1          0       0
186  0.419656  -0.324838 -0.631730  ...                 1          0       0
228 -0.743224   1.568690  0.155361  ...                 0          0       1
182 -0.977218   0.785161 -0.309738  ...                 0          1       0
236  0.646560   1.046337 -1.239936  ...                 1          1       1

[5 rows x 12 columns]
        Sales  CompPrice    Income  ...  ShelveLoc_Medium  Urban_Yes  US_Yes
18   2.273883  -0.977778  1.479105  ...                 0          0       1
372  0.107664  -0.259544 -0.667507  ...                 1          0       0
9   -0.994945   0.458691  1.586435  ...                 1          0       1
127 -0.346143   0.001632 -0.739060  ...                 1          1       1
379 -0.597864   0.001632  1.514881  ...                 0          1       0
```

2. ***Feature selection & Prediction: Develop a python program that performs a backward stepwise regression analysis and eliminate features using p-value of t-test (threshold 0.01). Hint: Drop one feature at a time while monitoring Adjusted R-squared [20 points]***
   a. ***Create a table that shows the process and justifies the one-by-one elimination. You need to display the AIC, BIC and Adjusted R2 as a predictive accuracy for each elimination inside the table. Display the eliminated and final selected features on the console. Display the p-value (of a feature to be eliminated) of the feature inside the table.***

⇨ **Code**

```
# 2. a

dependent_variable='Sales'
independent_variables=[col for col in df_train.columns if col != dependent_variable]

X_train=df_train[independent_variables]
X_test=df_test[independent_variables]
y_train=df_train[dependent_variable]
y_test=df_test[dependent_variable]

# creating an array for droped features
dropped_features=[]
# adding constant
X_model_train=sm.add_constant(X_train)
X_test=sm.add_constant(X_test)
# training Model
model=sm.OLS(y_train,X_model_train).fit()
print(model.summary())
# Creating PrettyTables
feature_names=X_model_train.columns
summary_table=PrettyTable()
summary_table.field_names=['process Update','AIC','BIC','Adj R^2','p-value']
compare_table=PrettyTable()
compare_table.field_names=['Model','R^2','Adj-R^2','AIC','BIC','MSE']
# adding initial summary to pretty table
p_value = model.pvalues['Population']
summary_table.add_row(['Population',round(model.aic,3),round(model.bic,3),round(model.rsquared_adj,3),round(p_value, 3)])
# Removing Population
X_model_train.drop(columns=['Population'],inplace=True)
dropped_features.append('Population')
model=sm.OLS(y_train,X_model_train).fit()
p_value = model.pvalues['Education']
summary_table.add_row(['Education',round(model.aic,3),round(model.bic,3),round(model.rsquared_adj,3),round(p_value, 3)])
print(model.summary())
```

```python
# adding initial summary to pretty table
p_value = model.pvalues['Population']
summary_table.add_row(['Population',round(model.aic,3),round(model.bic,3),round(model.rsquared_adj,3),round(p_value, 3)])
# Removing Population
X_model_train.drop(columns=['Population'],inplace=True)
dropped_features.append('Population')
model=sm.OLS(y_train,X_model_train).fit()
p_value = model.pvalues['Education']
summary_table.add_row(['Education',round(model.aic,3),round(model.bic,3),round(model.rsquared_adj,3),round(p_value, 3)])
print(model.summary())
# Removing Education
X_model_train.drop(columns=['Education'],inplace=True)
dropped_features.append('Education')
model=sm.OLS(y_train,X_model_train).fit()
p_value = model.pvalues['US_Yes']
summary_table.add_row(['US_Yes',round(model.aic,3),round(model.bic,3),round(model.rsquared_adj,3),round(p_value, 3)])
print(model.summary())
# Removing US_Yes
X_model_train.drop(columns=['US_Yes'],inplace=True)
dropped_features.append('US_Yes')
model=sm.OLS(y_train,X_model_train).fit()
p_value = model.pvalues['Urban_Yes']
summary_table.add_row(['Urban_Yes',round(model.aic,3),round(model.bic,3),round(model.rsquared_adj,3),round(p_value, 3)])
print(model.summary())
# Removing Urban_Yes
X_model_train.drop(columns=['Urban_Yes'],inplace=True)
dropped_features.append('Urban_Yes')
model=sm.OLS(y_train,X_model_train).fit()
print(model.summary())
print(summary_table)
print(f"The feature dropped{dropped_features}")
selected_features=X_model_train.columns
print(f"Features selected{selected_features}")
```

⇨ **Output**

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
+----------------+---------+---------+---------+---------+
| process Update |   AIC   |   BIC   | Adj R^2 | p-value |
+----------------+---------+---------+---------+---------+
|   Population   |  276.26 |  321.48 |  0.867  |  0.962  |
|   Education    | 274.262 | 315.714 |  0.868  |  0.599  |
|     US_Yes     | 272.549 | 310.233 |  0.868  |  0.338  |
|   Urban_Yes    | 271.499 | 305.414 |  0.868  |  0.213  |
+----------------+---------+---------+---------+---------+
The feature dropped['Population', 'Education', 'US_Yes', 'Urban_Yes']
Features selectedIndex(['const', 'CompPrice', 'Income', 'Advertising', 'Price', 'Age',
       'ShelveLoc_Good', 'ShelveLoc_Medium'],
      dtype='object')
                            OLS Regression Results
```

***b. Drop the insignificant features and perform a regression analysis OLS on the selected features. Take a screen shot of the OLS summary and place it here after each elimination.***

## ⇨ Output

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.872
Model:                            OLS   Adj. R-squared:                  0.867
Method:                 Least Squares   F-statistic:                     190.3
Date:                Sun, 15 Oct 2023   Prob (F-statistic):          3.54e-130
Time:                        22:36:55   Log-Likelihood:                -126.13
No. Observations:                 320   AIC:                             276.3
Df Residuals:                     308   BIC:                             321.5
Df Model:                          11
Covariance Type:            nonrobust
================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const              -0.7471      0.070    -10.744      0.000      -0.884      -0.610
CompPrice           0.5043      0.025     19.967      0.000       0.455       0.554
Income              0.1657      0.021      8.079      0.000       0.125       0.206
Advertising         0.2785      0.030      9.162      0.000       0.219       0.338
Population          0.0011      0.022      0.048      0.962      -0.042       0.044
Price              -0.7965      0.026    -31.204      0.000      -0.847      -0.746
Age                -0.2746      0.021    -13.214      0.000      -0.315      -0.234
Education          -0.0109      0.021     -0.520      0.603      -0.052       0.030
ShelveLoc_Good      1.7561      0.061     28.758      0.000       1.636       1.876
ShelveLoc_Medium    0.7021      0.050     13.976      0.000       0.603       0.801
Urban_Yes           0.0562      0.046      1.226      0.221      -0.034       0.146
US_Yes             -0.0605      0.061     -0.985      0.325      -0.181       0.060
==============================================================================
Omnibus:                        0.705   Durbin-Watson:                   2.057
Prob(Omnibus):                  0.703   Jarque-Bera (JB):                0.804
Skew:                           0.103   Prob(JB):                        0.669
Kurtosis:                       2.866   Cond. No.                         7.28
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.872
Model:                            OLS   Adj. R-squared:                  0.868
Method:                 Least Squares   F-statistic:                     210.0
Date:                Sun, 15 Oct 2023   Prob (F-statistic):          2.39e-131
Time:                        22:36:55   Log-Likelihood:                -126.13
No. Observations:                 320   AIC:                             274.3
Df Residuals:                     309   BIC:                             315.7
Df Model:                          10
Covariance Type:            nonrobust
================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const              -0.7466      0.069    -10.870      0.000      -0.882      -0.611
CompPrice           0.5042      0.025     20.025      0.000       0.455       0.554
Income              0.1656      0.020      8.105      0.000       0.125       0.206
Advertising         0.2789      0.029      9.621      0.000       0.222       0.336
Price              -0.7965      0.025    -31.264      0.000      -0.847      -0.746
Age                -0.2746      0.021    -13.241      0.000      -0.315      -0.234
Education          -0.0110      0.021     -0.527      0.599      -0.052       0.030
ShelveLoc_Good      1.7559      0.061     28.840      0.000       1.636       1.876
ShelveLoc_Medium    0.7020      0.050     14.031      0.000       0.604       0.800
Urban_Yes           0.0560      0.046      1.227      0.221      -0.034       0.146
US_Yes             -0.0609      0.061     -1.005      0.316      -0.180       0.058
==============================================================================
Omnibus:                        0.711   Durbin-Watson:                   2.056
Prob(Omnibus):                  0.701   Jarque-Bera (JB):                0.810
Skew:                           0.104   Prob(JB):                        0.667
```

```
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.872
Model:                            OLS   Adj. R-squared:                  0.868
Method:                 Least Squares   F-statistic:                     233.8
Date:                Sun, 15 Oct 2023   Prob (F-statistic):          1.74e-132
Time:                        22:36:55   Log-Likelihood:                -126.27
No. Observations:                 320   AIC:                             272.5
Df Residuals:                     310   BIC:                             310.2
Df Model:                           9
Covariance Type:            nonrobust
================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const              -0.7496      0.068    -10.960      0.000      -0.884      -0.615
CompPrice           0.5043      0.025     20.052      0.000       0.455       0.554
Income              0.1661      0.020      8.150      0.000       0.126       0.206
Advertising         0.2780      0.029      9.618      0.000       0.221       0.335
Price              -0.7972      0.025    -31.376      0.000      -0.847      -0.747
Age                -0.2746      0.021    -13.257      0.000      -0.315      -0.234
ShelveLoc_Good      1.7572      0.061     28.915      0.000       1.638       1.877
ShelveLoc_Medium    0.7023      0.050     14.055      0.000       0.604       0.801
Urban_Yes           0.0566      0.046      1.242      0.215      -0.033       0.146
US_Yes             -0.0579      0.060     -0.960      0.338      -0.176       0.061
==============================================================================
Omnibus:                        0.760   Durbin-Watson:                   2.054
Prob(Omnibus):                  0.684   Jarque-Bera (JB):                0.877
Skew:                           0.092   Prob(JB):                        0.645
Kurtosis:                       2.821   Cond. No.                         7.15
==============================================================================
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.871
Model:                            OLS   Adj. R-squared:                  0.868
Method:                 Least Squares   F-statistic:                     263.0
Date:                Sun, 15 Oct 2023   Prob (F-statistic):          1.64e-133
Time:                        22:36:55   Log-Likelihood:                -126.75
No. Observations:                 320   AIC:                             271.5
Df Residuals:                     311   BIC:                             305.4
Df Model:                           8
Covariance Type:            nonrobust
================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const              -0.7894      0.054    -14.518      0.000      -0.896      -0.682
CompPrice           0.5038      0.025     20.038      0.000       0.454       0.553
Income              0.1652      0.020      8.114      0.000       0.125       0.205
Advertising         0.2588      0.021     12.405      0.000       0.218       0.300
Price              -0.7982      0.025    -31.443      0.000      -0.848      -0.748
Age                -0.2749      0.021    -13.276      0.000      -0.316      -0.234
ShelveLoc_Good      1.7575      0.061     28.924      0.000       1.638       1.877
ShelveLoc_Medium    0.7064      0.050     14.193      0.000       0.609       0.804
Urban_Yes           0.0568      0.046      1.247      0.213      -0.033       0.146
==============================================================================
Omnibus:                        0.758   Durbin-Watson:                   2.047
Prob(Omnibus):                  0.685   Jarque-Bera (JB):                0.867
Skew:                           0.102   Prob(JB):                        0.648
Kurtosis:                       2.848   Cond. No.                         5.72
==============================================================================
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.871
Model:                            OLS   Adj. R-squared:                  0.868
Method:                 Least Squares   F-statistic:                     299.8
Date:                Sun, 15 Oct 2023   Prob (F-statistic):          1.98e-134
Time:                        22:36:55   Log-Likelihood:                -127.55
No. Observations:                 320   AIC:                             271.1
Df Residuals:                     312   BIC:                             301.2
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.7447      0.041    -18.183      0.000      -0.825      -0.664
CompPrice      0.5057      0.025     20.136      0.000       0.456       0.555
Income         0.1663      0.020      8.173      0.000       0.126       0.206
Advertising    0.2586      0.021     12.387      0.000       0.218       0.300
Price         -0.7986      0.025    -31.434      0.000      -0.849      -0.749
Age           -0.2740      0.021    -13.228      0.000      -0.315      -0.233
ShelveLoc_Good  1.7521     0.061     28.882      0.000       1.633       1.872
ShelveLoc_Medium  0.7011   0.050     14.126      0.000       0.603       0.799
==============================================================================
Omnibus:                        0.862   Durbin-Watson:                   2.053
Prob(Omnibus):                  0.650   Jarque-Bera (JB):                0.976
Skew:                           0.099   Prob(JB):                        0.614
Kurtosis:                       2.815   Cond. No.                         4.91
==============================================================================
```

```
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.872
Model:                            OLS   Adj. R-squared:                  0.867
Method:                 Least Squares   F-statistic:                     190.3
Date:                Sun, 15 Oct 2023   Prob (F-statistic):          3.54e-130
Time:                        21:58:49   Log-Likelihood:                -126.13
No. Observations:                 320   AIC:                             276.3
Df Residuals:                     308   BIC:                             321.5
Df Model:                          11
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.7471      0.070    -10.744      0.000      -0.884      -0.610
CompPrice      0.5043      0.025     19.967      0.000       0.455       0.554
Income         0.1657      0.021      8.079      0.000       0.125       0.206
Advertising    0.2785      0.030      9.162      0.000       0.219       0.338
Population     0.0011      0.022      0.048      0.962      -0.042       0.044
Price         -0.7965      0.026    -31.204      0.000      -0.847      -0.746
Age           -0.2746      0.021    -13.214      0.000      -0.315      -0.234
Education     -0.0109      0.021     -0.520      0.603      -0.052       0.030
ShelveLoc_Good  1.7561     0.061     28.758      0.000       1.636       1.876
ShelveLoc_Medium  0.7021   0.050     13.976      0.000       0.603       0.801
US_Yes        -0.0605      0.061     -0.985      0.325      -0.181       0.060
Urban_Yes      0.0562      0.046      1.226      0.221      -0.034       0.146
==============================================================================
Omnibus:                        0.705   Durbin-Watson:                   2.057
Prob(Omnibus):                  0.703   Jarque-Bera (JB):                0.804
Skew:                           0.103   Prob(JB):                        0.669
Kurtosis:                       2.866   Cond. No.                         7.28
==============================================================================
```

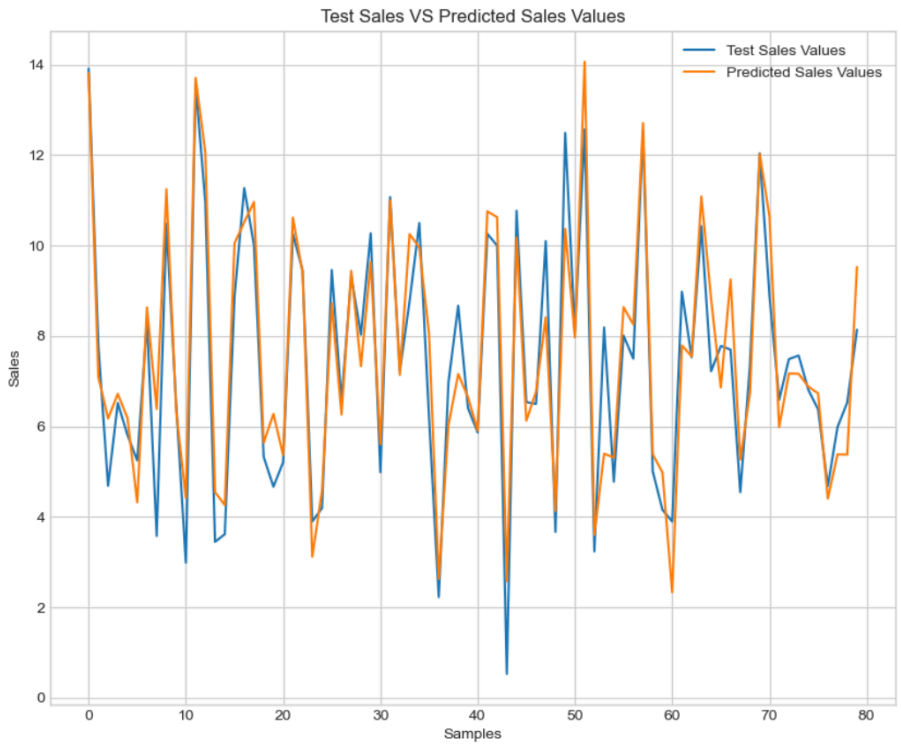
***c. After removing insignificant features, write the final regression model. Make a prediction on sales and compare it with the test set and plot the original test set***

***(sales without transformation) versus the predicted values (not a scatter plot). Hint: You need to perform a reverse transformation.***

⇨ **Code**

```
# 2. c

final_features=X_model_train.columns
X_model_test=X_test[final_features]
y_pred=model.predict(X_model_test)

# Destandardizing the  y_pred and y_test Values
y_scaler=StandardScaler()
y_scaler.fit_transform(df[['Sales']])
y_test_destand=y_scaler.inverse_transform(y_test.values.reshape(-1,1)).flatten()
y_pred_2d=y_pred.values.reshape(-1,1)
y_pred_destand=y_scaler.inverse_transform(y_pred_2d)

# 2. d
mse_1 = mean_squared_error(y_test_destand,y_pred_destand)
print(f"Mean Squared Error (MSE): {mse_1:.3f} ")
```

⇨ **Output**



**d. Display the Mean Squared Error (MSE) on this prediction on the console.**

⇨ **Code**

```
# 2. d
mse_1 = mean_squared_error(y_test_destand,y_pred_destand)
print(f"Mean Squared Error (MSE): {mse_1:.3f} ")
```

⇨ **Output**

```
Final Regression Equation:
Y = -0.745 + 0.506 * CompPrice + 0.166 * Income + 0.259 * Advertising + -0.799 * Price + -0.274 * Age + 1.752 * ShelveLoc_Good + 0.701 * ShelveLoc_Medium
Mean Squared Error (MSE): 0.984
```

***3. Feature selection & Prediction: Develop a python program that performs a Principal Component Analysis (PCA). [15 points]***

***a. How many features are needed that explain more than 90% of the dependent variance.***

⇨ **Code**

```
# 3a. How many features are needed that explain more than 90% of the dependent variance.
explained_var_ratio = pca.explained_variance_ratio_
cumulative_var_ratio = np.cumsum(explained_var_ratio)
num_features_90_percent = np.where(cumulative_var_ratio > 0.9)[0][0] + 1
print("Number of features needed to explain more than 90% of the dependent variance:", num_features_90_percent)

# 3b. Plot the cumulative explained variance versus the number of features.
plt.figure(figsize=(10, 5))
plt.plot( *args: np.arange(1, len(cumulative_var_ratio) + 1, 1), cumulative_var_ratio, label="Cumulative Explained Variance")
plt.xticks(np.arange(1, len(cumulative_var_ratio) + 1, 1))
plt.xlabel("Number of Features")
plt.ylabel("Cumulative Explained Variance")
plt.grid(True)
plt.legend()
plt.title("Cumulative Explained Variance vs. Number of Features")
plt.show()

# 3c. Draw a vertical line and horizontal line showing the exact 90% threshold and the corresponding number of features.
plt.figure(figsize=(10, 5))
plt.plot( *args: np.arange(1, len(cumulative_var_ratio) + 1, 1), cumulative_var_ratio, label="Cumulative Explained Variance")
plt.axvline(x=num_features_90_percent, color="r", linestyle="--", label="90% Threshold")
plt.axhline(y=0.9, color="r", linestyle="--")
plt.xticks(np.arange(1, len(cumulative_var_ratio) + 1, 1))
plt.xlabel("Number of Features")
plt.ylabel("Cumulative Explained Variance")
plt.grid(True)
plt.legend()
plt.title("Cumulative Explained Variance vs. Number of Features with 90% Threshold")
plt.show()
```

⇨ **Output**

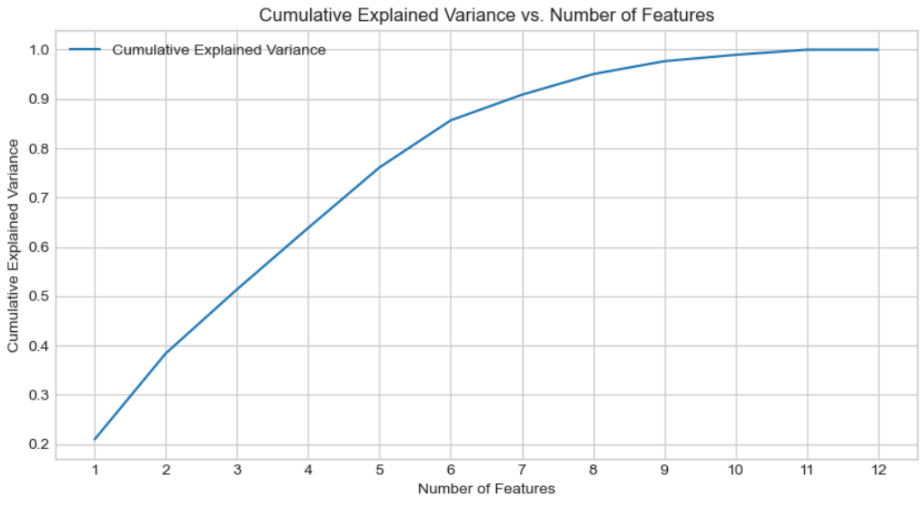
```
Number of components needed to explain 90% of the variance: 7
```

***b. Plot the cumulative explained variance versus the number of features. Hint: The number of features lie on the x-axis and must start off at 1.***

⇨ **Code**

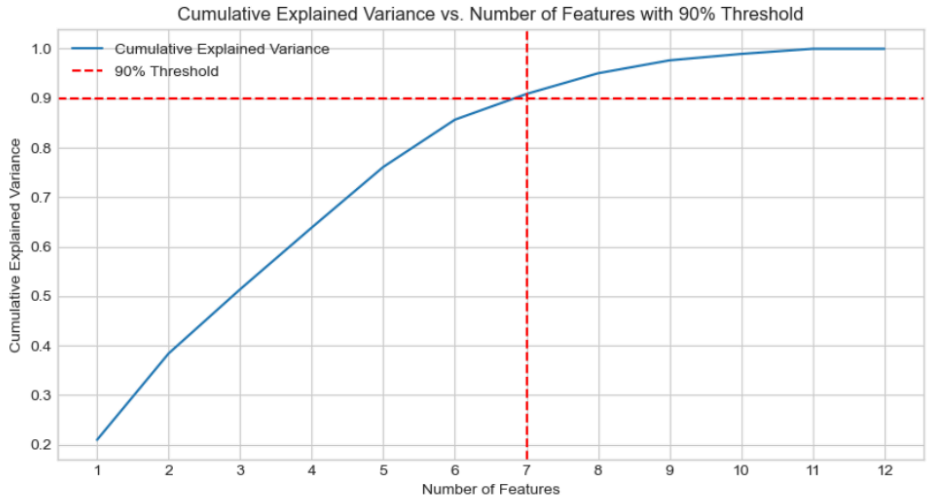
```
# 3b. Plot the cumulative explained variance versus the number of features.
plt.figure(figsize=(10, 5))
plt.plot( *args: np.arange(1, len(cumulative_var_ratio) + 1, 1), cumulative_var_ratio, label="Cumulative Explained Variance")
plt.xticks(np.arange(1, len(cumulative_var_ratio) + 1, 1))
plt.xlabel("Number of Features")
plt.ylabel("Cumulative Explained Variance")
plt.grid(True)
plt.legend()
plt.title("Cumulative Explained Variance vs. Number of Features")
plt.show()
```

⇨ **Output**




**c. Draw a vertical line and horizontal line showing the exact 90% threshold and the corresponding number of features.**

⇨ **Output**




**d. What does the PCA say? Do the PCA tell you which feature(s) need to remove? Explain your answer.**

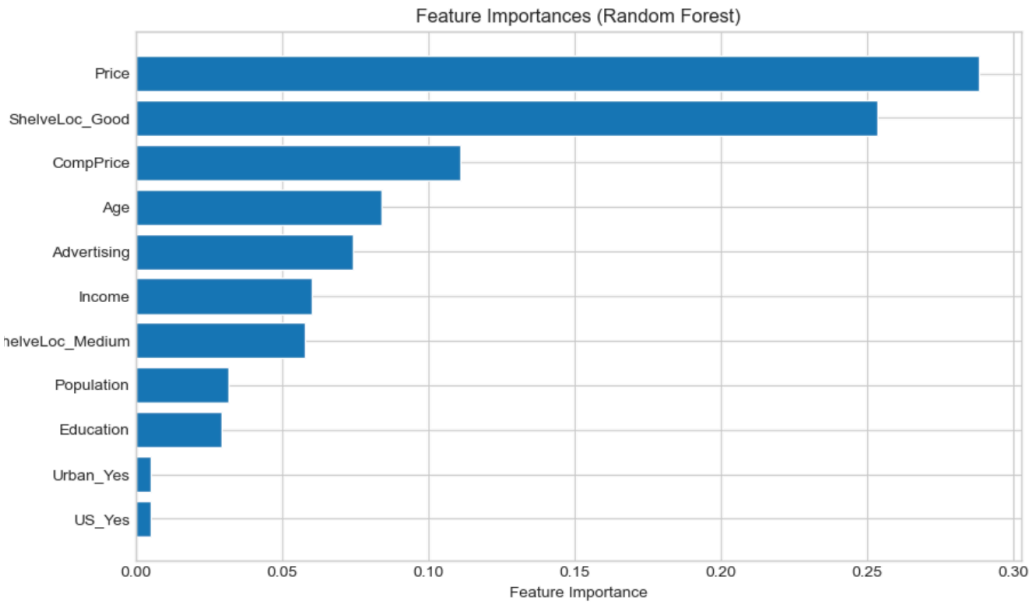
- ⇨ PCA does not tell us which features need to be removed, it just provides us with insights into the relationship between features and their importance explaining the variance between the data.
- ⇨ PCA provides insights into feature correlation. Features that are highly correlated may have a strong impact on the principle components.
- ⇨ However, PCA does not provide explicit guidance on which features should be removed for a specific problem. It's a tool to help you make informed decisions about feature selection, but the final choice depends on the problem context and modeling goals.

4. ***Feature selection & Prediction: Develop a python program that performs a Random Forest Analysis and eliminates features based on the importance. [25 points]***
   a. ***Plot the horizontal bar graph (once random forest analysis is complete) that displays the features importance in descending order. Hint: Make sure that the y-axis is labeled according to the feature labels.***

⇨ **Code:**

```
# 4. a
X_rf=df_one_hot_encode.drop(columns=[dependent_variable])
y_rf=df[dependent_variable]
rf_model = RandomForestRegressor(n_estimators=100, random_state=5805)
rf_model.fit(X_rf, y_rf)
feature_importances = rf_model.feature_importances_
feature_labels = X_rf.columns
### sorting feature Importances ###
sorted_indices = np.argsort(feature_importances)[::-1]
sorted_importances = feature_importances[sorted_indices]
sorted_labels = feature_labels[sorted_indices]
### plot the feature importance ##
plt.figure(figsize=(10,6))
plt.barh(range(len(sorted_labels)),sorted_importances,align='center')
plt.yticks(range(len(sorted_labels)),sorted_labels)
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Feature Importances (Random Forest)')
plt.gca().invert_yaxis()

plt.show()
```

⇨ **Output**




⇨

***b. Display the eliminated and final selected features on the console. Does the selected features base on random forest and stepwise regression method identical? Explain your answer.***

⇨ **Output**

```
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
The dropped features are ['US_Yes', 'Urban_Yes']
The final Selected features are :Index(['const', 'CompPrice', 'Income', 'Advertising', 'Population', 'Price',
       'Age', 'Education', 'ShelveLoc_Good', 'ShelveLoc_Medium'],
      dtype='object')
Mean Squared Error (MSE): 0.974
```

⇨ **Explanation:** The selected features based on the Random Forest method and the Stepwise Regression method are not necessarily identical, and whether they are the same or different depends on the data and the specific goals of the feature selection process.

⇨ Random Forest assesses feature importance by measuring their contributions to reducing uncertainty in decision tree construction, potentially ranking features with non-linear relationships higher. It excels at capturing intricate feature connections to the target.

⇨ Stepwise Regression, a classical statistical approach, selects or excludes features using statistical criteria, typically p-values or information criteria like AIC/BIC. Its primary aim is to enhance model fit through statistically significant features, often favoring those with linear relationships to the target.

***c. Drop the insignificant features (pick a threshold) and perform a regression analysis OLS on the selected features. Take a screen shot of the OLS summary and place it here.***

⇨ **Code:**

```
# 4 c
# Selecting Threshold as 0.02 and dropping Urban_Yes and US_Yes
X_model_train_rf=sm.add_constant(X_train)
X_test=sm.add_constant(X_test)
dropped_features_rfs=[]


X_model_train_rf.drop(columns=['US_Yes'],inplace=True)
dropped_features_rfs.append('US_Yes')
model=sm.OLS(y_train,X_model_train_rf).fit()
X_model_train_rf.drop(columns=['Urban_Yes'],inplace=True)
dropped_features_rfs.append('Urban_Yes')
model=sm.OLS(y_train,X_model_train_rf).fit()
print(model.summary())
```

⇨ **Output**

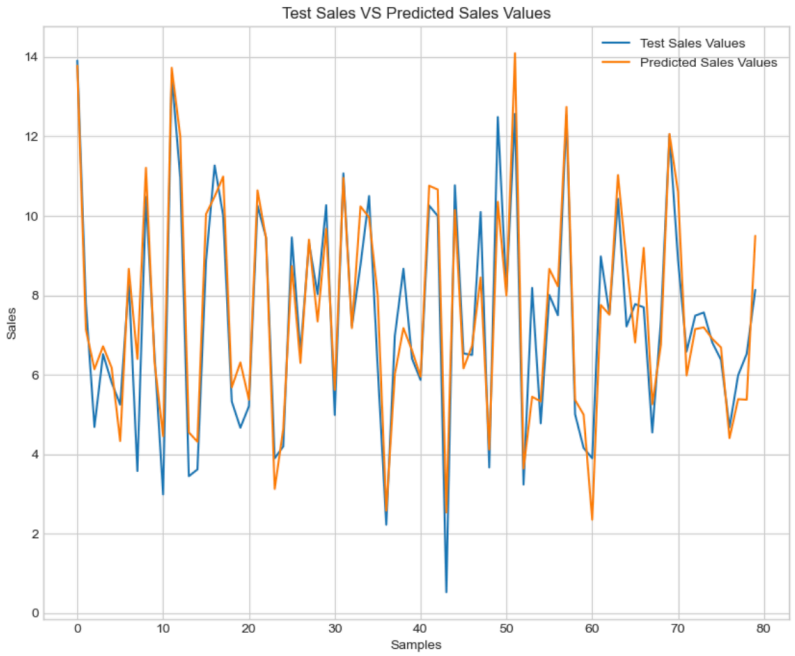
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.871
Model:                            OLS   Adj. R-squared:                  0.867
Method:                 Least Squares   F-statistic:                     231.9
Date:                Sun, 15 Oct 2023   Prob (F-statistic):          5.31e-132
Time:                        23:01:29   Log-Likelihood:                -127.43
No. Observations:                 320   AIC:                             274.9
Df Residuals:                     310   BIC:                             312.5
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const             -0.7446      0.041    -18.117      0.000      -0.825      -0.664
CompPrice          0.5057      0.025     20.043      0.000       0.456       0.555
Income             0.1660      0.020      8.100      0.000       0.126       0.206
Advertising        0.2579      0.022     11.860      0.000       0.215       0.301
Population         0.0025      0.022      0.113      0.910      -0.040       0.045
Price             -0.7981      0.026    -31.276      0.000      -0.848      -0.748
Age               -0.2739      0.021    -13.182      0.000      -0.315      -0.233
Education         -0.0095      0.021     -0.453      0.651      -0.051       0.032
ShelveLoc_Good     1.7514      0.061     28.742      0.000       1.632       1.871
ShelveLoc_Medium   0.7013      0.050     14.072      0.000       0.603       0.799
==============================================================================
Omnibus:                        0.804   Durbin-Watson:                   2.057
Prob(Omnibus):                  0.669   Jarque-Bera (JB):                0.910
Skew:                           0.107   Prob(JB):                        0.635
Kurtosis:                       2.851   Cond. No.                         4.93
==============================================================================
```

***d. Make a prediction using the test set and plot the original test set (sales without transformation) versus the predicted values (not a scatter plot). Hint: You need to perform a reverse transformation.***

⇨ **Code**

```python
# 4. d
final_features_rf=X_model_train_rf.columns
X_model_test_rf=X_test[final_features_rf]
y_pred_rf=model.predict(X_model_test_rf)
# Destandardizing the  y_pred and y_test Values
y_scaler=StandardScaler()
y_scaler.fit_transform(df[['Sales']])
y_test_destand=y_scaler.inverse_transform(y_test.values.reshape(-1,1)).flatten()
y_pred_2d_rf=y_pred_rf.values.reshape(-1,1)
y_pred_destand_rf=y_scaler.inverse_transform(y_pred_2d_rf)

# plot graph between y_pred_destand and y_test_destand

plt.figure(figsize=(10,8))
plt.plot( *args: y_test_destand,label='Test Sales Values')
plt.plot( *args: y_pred_destand_rf,label='Predicted Sales Values')
plt.xlabel('Samples')
plt.ylabel('Sales')
plt.title('Test Sales VS Predicted Sales Values')
plt.grid(True)
plt.legend()
plt.show()

# 4e
mse_2 = mean_squared_error(y_test_destand,y_pred_destand_rf)
print(f"Mean Squared Error (MSE): {mse_2:.3f} ")

#=============================================================================================================
```

⇨ **Output**




*e. Display the Mean Squared Error (MSE) on this prediction on the console.*

⇨ **Code**

```
# 4. d
final_features_rf=X_model_train_rf.columns
X_model_test_rf=X_test[final_features_rf]
y_pred_rf=model.predict(X_model_test_rf)
# Destandardizing the  y_pred and y_test Values
y_scaler=StandardScaler()
y_scaler.fit_transform(df[['Sales']])
y_test_destand=y_scaler.inverse_transform(y_test.values.reshape(-1,1)).flatten()
y_pred_2d_rf=y_pred_rf.values.reshape(-1,1)
y_pred_destand_rf=y_scaler.inverse_transform(y_pred_2d_rf)

# plot graph between y_pred_destand and y_test_destand

plt.figure(figsize=(10,8))
plt.plot( *args: y_test_destand,label='Test Sales Values')
plt.plot( *args: y_pred_destand_rf,label='Predicted Sales Values')
plt.xlabel('Samples')
plt.ylabel('Sales')
plt.title('Test Sales VS Predicted Sales Values')
plt.grid(True)
plt.legend()
plt.show()

# 4e
mse_2 = mean_squared_error(y_test_destand,y_pred_destand_rf)
print(f"Mean Squared Error (MSE): {mse_2:.3f} ")

#=============================================================================================================
```

⇨ **Output**

```
Mean Squared Error (MSE): 0.974
```

*5. Create a table that compares the R-squared, Adjusted R-squared, AIC, BIC and MSE of the prediction in step 2 and 4. Which method of feature selection will you recommend for this problem and what features do you recommend for elimination? Explain your answer.*

⇨ **Code**

```
print(compare_table)
```

⇨ **Output**

```
+---------------------+-------+---------+---------+---------+-------+
|        Model        |  R^2  | Adj-R^2 |   AIC   |   BIC   |  MSE  |
+---------------------+-------+---------+---------+---------+-------+
| Backward Regression | 0.871 |  0.868  | 271.094 | 301.241 | 0.984 |
|    Random Forest    | 0.871 |  0.867  | 274.861 | 312.545 | 0.974 |
+---------------------+-------+---------+---------+---------+-------+
```

⇨ ***Explanation:***

⇨ Adjusted R-squared: This tells us how good a model is at making predictions. When it's higher, it means the model is doing a bit better. It also looks at how many things we use to make predictions and doesn't like it when we use things that don't help much. So, if the number is higher, the model is better, but not too complicated.

⇨ AIC This is like a score for models. Lower is better. It looks at how well the model fits the data but also checks if it's too complicated. So, if the number is lower, it's a better model. For example, for the Backward Regression model, the AIC is 271.094, and for the Random Forest model, it's 274.861.

⇨ BIC It's a bit like AIC. Lower is better. It also looks at how well the model fits the data and keeps an eye on the model's complexity. For instance, the BIC for the Backward Regression model is 301.241, and for the Random Forest model, it's 312.545.

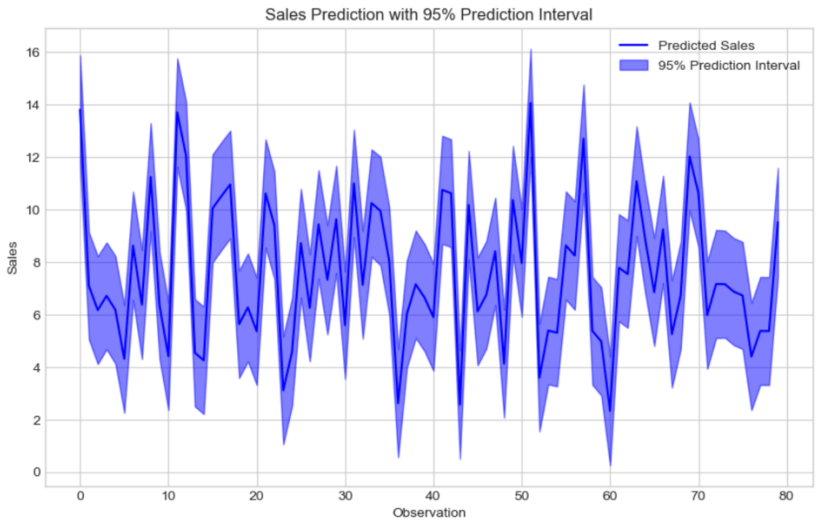
⇨ MSE: This tells us how close our model's predictions are to the real data. Lower is better because it means our predictions are really close to the real stuff. For the Backward Regression model, the MSE is 0.984, while for the Random Forest model, it's 0.974.

6. **Prediction Interval: Based on the answer in the previous question and the selected features using stepwise regression, find the prediction intervals for the regression model. Plot the predicted sales value (need to perform reverse transformation), the lower and upper prediction intervals (95%) in one graph. Hint: You can use .get_prediction() function. [10 points]**

   ⇨ **Code**

```
predictions = model.predict(X_model_test)
pred_int = model.get_prediction(X_model_test).summary_frame(alpha=0.05)
predictions_original=y_scaler.inverse_transform(predictions.values.reshape(-1,1)).flatten()
# Extract the lower and upper prediction interval bounds
lower_pred = pred_int['obs_ci_lower']
upper_pred = pred_int['obs_ci_upper']
lower_pred_orig=y_scaler.inverse_transform(lower_pred.values.reshape(-1,1)).flatten()
upper_pred_orig=y_scaler.inverse_transform(upper_pred.values.reshape(-1,1)).flatten()

# Reverse any transformations on the predictions if needed
# For example, if you've transformed the target variable during modeling, you may need to reverse the transform

# Create a plot of predictions
plt.figure(figsize=(10, 6))
plt.plot( *args: predictions_original, label="Predicted Sales", color='red')
plt.fill_between(np.arange(len(predictions)), lower_pred_orig, upper_pred_orig, color='gray', alpha=0.5,
                 label="95% Prediction Interval")
plt.legend()
plt.xlabel("Observation")
plt.ylabel("Sales")
plt.title("Sales Prediction with 95% Prediction Interval")
plt.grid(True)
plt.show()
### plot graph between y_pred_destand and y_test_destand ######

plt.figure(figsize=(10,8))
plt.plot( *args: y_test_destand,label='Test Sales Values')
plt.plot( *args: y_pred_destand,label='Predicted Sales Values')
plt.xlabel('Samples')
plt.ylabel('Sales')
plt.title('Test Sales VS Predicted Sales Values')
plt.grid(True)
plt.legend()
```

   ⇨ **Output**



7. **Polynomial regression and Grid Search: Let suppose we want to find a polynomial regression model that regresses Sales (dependent variable) versus the "price" feature only. Hint: You may use the following packages: [25 points]**

   a. **Perform a grid search that minimized RMSE.**

      ⇨ **Code**

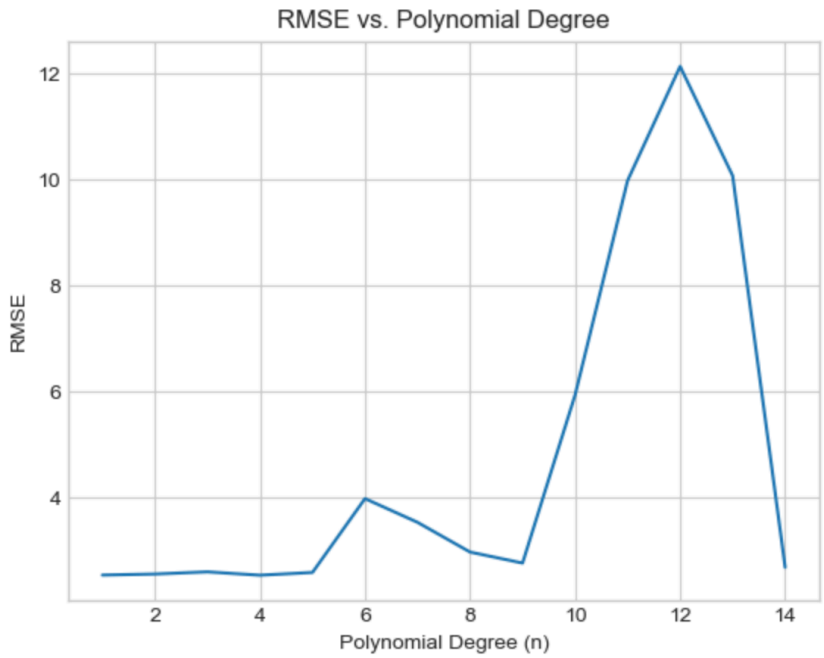
```
#7. a

X_pr_price = df['Price']
y_pr = df['Sales']
X_pr_price = X_pr_price.values.reshape(-1, 1)
y_pr=y_pr.values.reshape(-1,1)
param_grid = {'polynomialfeatures__degree': np.arange(1, 15)}

model = make_pipeline( *steps: PolynomialFeatures(), LinearRegression())

grid_search = GridSearchCV(model, param_grid, scoring='neg_mean_squared_error', cv=5)
grid_search.fit(np.array(X_pr_price).reshape(-1,1),np.array(y_pr).reshape(-1,1))

# 7. b
best_degree = grid_search.best_params_['polynomialfeatures__degree']
print("Optimum Order (n):", best_degree)

# 7. c
results = grid_search.cv_results_
rmse = np.sqrt(-results['mean_test_score'])
plt.plot( *args: range(1, 15), rmse)
plt.xlabel('Polynomial Degree (n)')
plt.ylabel('RMSE')
plt.title('RMSE vs. Polynomial Degree')
plt.grid(True)
plt.show()
```

   b. **What is the optimum order for n?**

      ⇨ **Code**

```
#7. a

X_pr_price = df['Price']
y_pr = df['Sales']
X_pr_price = X_pr_price.values.reshape(-1, 1)
y_pr=y_pr.values.reshape(-1,1)
param_grid = {'polynomialfeatures__degree': np.arange(1, 15)}

model = make_pipeline( *steps: PolynomialFeatures(), LinearRegression())

grid_search = GridSearchCV(model, param_grid, scoring='neg_mean_squared_error', cv=5)
grid_search.fit(np.array(X_pr_price).reshape(-1,1),np.array(y_pr).reshape(-1,1))

# 7. b
best_degree = grid_search.best_params_['polynomialfeatures__degree']
print("Optimum Order (n):", best_degree)

# 7. c
results = grid_search.cv_results_
rmse = np.sqrt(-results['mean_test_score'])
plt.plot( *args: range(1, 15), rmse)
plt.xlabel('Polynomial Degree (n)')
plt.ylabel('RMSE')
plt.title('RMSE vs. Polynomial Degree')
plt.grid(True)
plt.show()
```

      ⇨ **Output**



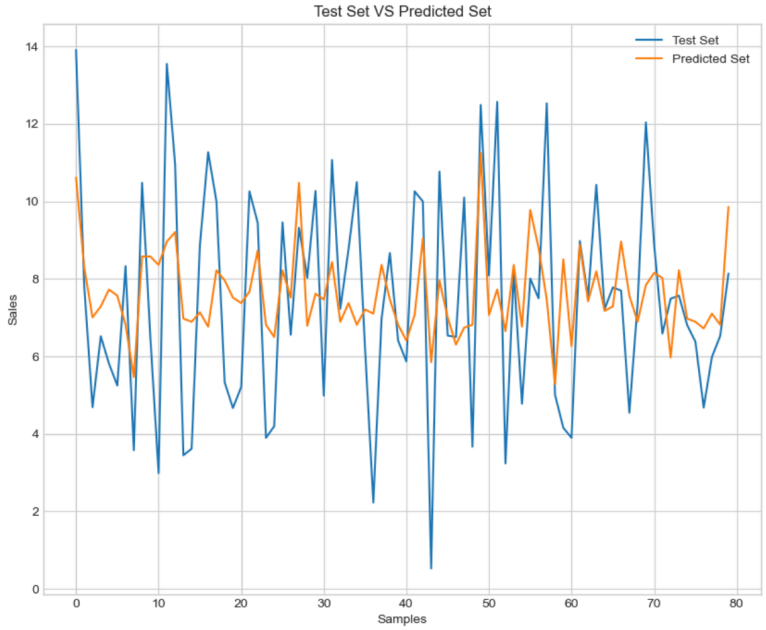

```
Optimum Order (n): 4
Mean Squared Error (MSE): 5.826
```

**c. Plot the RMSE versus the n order. The search intervals [1,15]. d. Split the dataset into train-test 80-20 random_state =5805. Train the regression model using OLS and the optimum nth order derived from the previous section. Plot the test set (sales) versus the predicted sales values.**

⇨ **Code**

```
#7. a

X_pr_price = df['Price']
y_pr = df['Sales']
X_pr_price = X_pr_price.values.reshape(-1, 1)
y_pr=y_pr.values.reshape(-1,1)
param_grid = {'polynomialfeatures__degree': np.arange(1, 15)}

model = make_pipeline( *steps: PolynomialFeatures(), LinearRegression())

grid_search = GridSearchCV(model, param_grid, scoring='neg_mean_squared_error', cv=5)
grid_search.fit(np.array(X_pr_price).reshape(-1,1),np.array(y_pr).reshape(-1,1))

# 7. b
best_degree = grid_search.best_params_['polynomialfeatures__degree']
print("Optimum Order (n):", best_degree)

# 7. c
results = grid_search.cv_results_
rmse = np.sqrt(-results['mean_test_score'])
plt.plot( *args: range(1, 15), rmse)
plt.xlabel('Polynomial Degree (n)')
plt.ylabel('RMSE')
plt.title('RMSE vs. Polynomial Degree')
plt.grid(True)
plt.show()
```

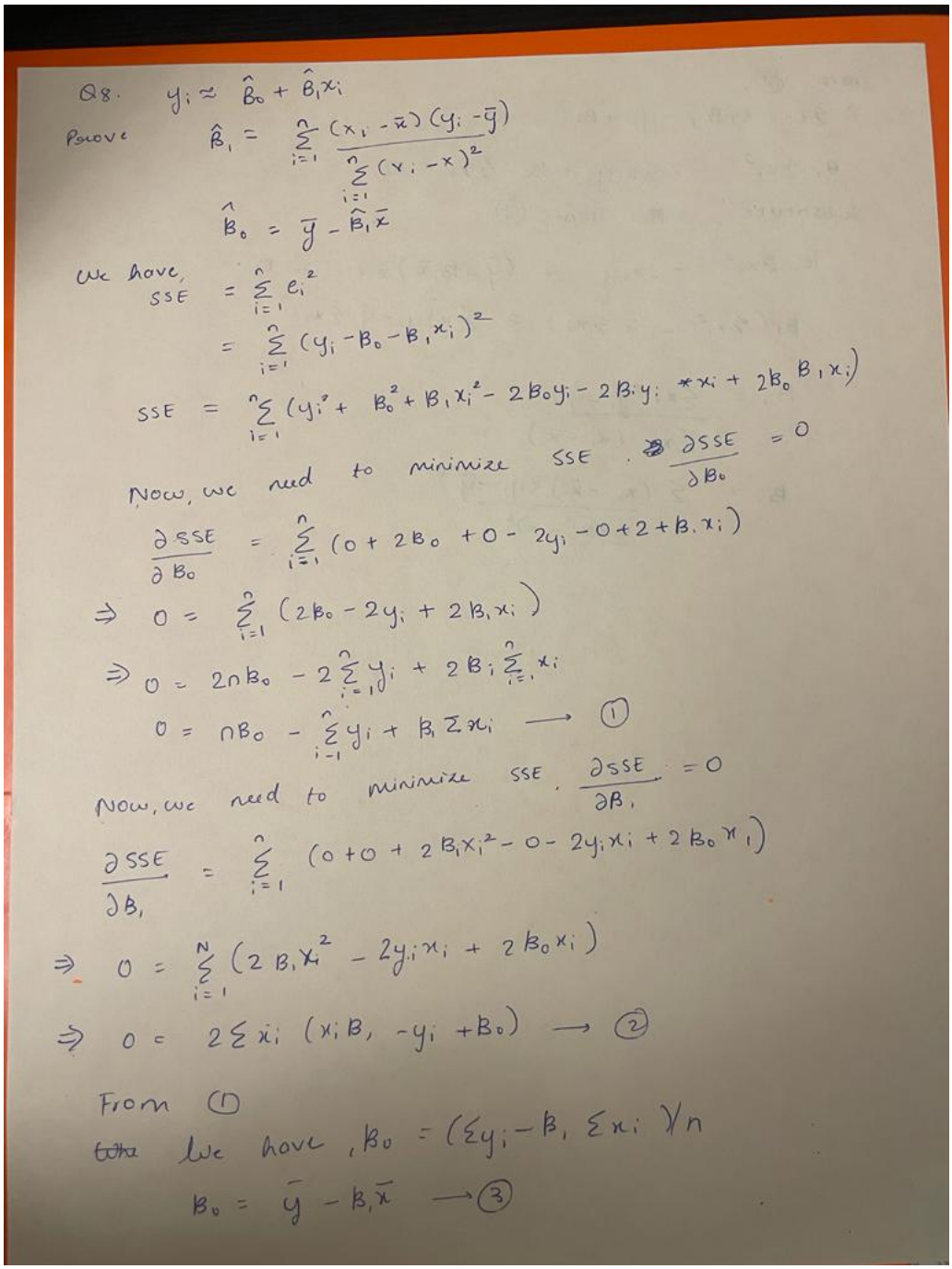
⇨ **Output**




**d. What is the MSE of this nth other polynomial regression prediction?**

```
Optimum Order (n): 4
Mean Squared Error (MSE): 5.826
```

8. ***In a simple linear regression with n observations. Hint: You need to construct the sum of squared of error function and minimize the function with respect to beta_0 and beta_1. This requires a derivative. [10 points] Prove the following:***



Q8. $y_i \simeq \hat{B}_0 + \hat{B}_1 x_i$

Prove $\hat{B}_1 = \dfrac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}$

$\hat{B}_0 = \bar{y} - \hat{B}_1 \bar{x}$

We have, $SSE = \sum_{i=1}^{n} e_i^2$

$= \sum_{i=1}^{n} (y_i - B_0 - B_1 x_i)^2$

$SSE = \sum_{i=1}^{n} (y_i^2 + B_0^2 + B_1 x_i^2 - 2B_0 y_i - 2B_1 y_i * x_i + 2B_0 B_1 x_i)$

Now, we need to minimize SSE. $\dfrac{\partial SSE}{\partial B_0} = 0$

$\dfrac{\partial SSE}{\partial B_0} = \sum_{i=1}^{n} (0 + 2B_0 + 0 - 2y_i - 0 + 2 * B_1 x_i)$

$\Rightarrow 0 = \sum_{i=1}^{n} (2B_0 - 2y_i + 2B_1 x_i)$

$\Rightarrow 0 = 2nB_0 - 2\sum_{i=1}^{n} y_i + 2B_1 \sum_{i=1}^{n} x_i$

$0 = nB_0 - \sum_{i=1}^{n} y_i + B_1 \Sigma x_i \longrightarrow (1)$

Now, we need to minimize SSE. $\dfrac{\partial SSE}{\partial B_1} = 0$

$\dfrac{\partial SSE}{\partial B_1} = \sum_{i=1}^{n} (0 + 0 + 2B_1 x_i^2 - 0 - 2y_i x_i + 2B_0 x_i)$

$\Rightarrow 0 = \sum_{i=1}^{N} (2B_1 x_i^2 - 2y_i x_i + 2B_0 x_i)$

$\Rightarrow 0 = 2\Sigma x_i (x_i B_1 - y_i + B_0) \longrightarrow (2)$

From (1)

We have, $B_0 = (\Sigma y_i - B_1 \Sigma x_i)/n$

$B_0 = \bar{y} - B_1 \bar{x} \longrightarrow (3)$

From ②,

$$2 \sum x_i (x_i B_1 - y_i + B_0) = 0$$

$$B_1 \sum x_i^2 - \sum x_i y_i + B_0 \sum x_i = 0.$$

Substitute $B_0$, from ③

$$B_1 \sum x_i^2 - x_i y_i + (\bar{y} - B_1 \bar{x}) \sum x_i = 0$$

$$B_1 \left( \sum x_i^2 - \bar{x} \sum x_i \right) = \sum x_i y_i - \bar{y} \sum x_i$$

$$B_1 = \frac{\sum x_i (y_i - \bar{y})}{\sum x_i (x_i - \bar{x})}$$

$$B_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - x)^2}.$$