AI5213/EC4213: Machine Learning and Deep Learning **Student ID: 20195024**
Homework 4 (Beyond Linearity, Tree Model) **Name: Kim, Minjun**

Due: Sunday, Nov 5, 11:59PM

This homework comprises a set of conceptual problems and coding exercises. Some problems are trivial, while others will require a lot of thought. Start this homework early!

> Guideline for those new to data analysis using Python:
> We recommend you to review the Lab section within each chapter (e.g., p. 310 of Ch. 7 or https://islp.readthedocs.io/en/latest/labs/Ch08-baggboost-lab.html) prior to tackling the programming tasks. Additionally, find datasets and Jupyter notebooks at https://github.com/intro-stat-learning/ISLP_labs/ and https://islp.readthedocs.io/en/latest/.
>
> Visit https://www.statlearning.com/forum — a dedicated forum created by and for the ISL community. Whether you have a question or encounter issues with ISLP labs, this platform is your go-to resource for assistance and collaborative discussions.

Deliverables:

1. Submit a PDF of your homework, with an appendix listing all your code, to the Gradescope assignment entitled "HW4 Write-Up". You may typeset your homework in LaTeX or Word or submit neatly handwritten and scanned solutions. Please start each question on a new page. If there are graphs, include those graphs in the correct sections. **Take a screenshot of the code and attach to each question.** Do not put them in an appendix. We need each solution to be self-contained on pages of its own.

   • On the first page of your write-up, please sign your signature next to the following statement. (Mac Preview, PDF Expert, and Foxit PDF Reader, among others, have tools to let you sign a PDF file.) We want to make extra clear the consequences of cheating.

   "I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted."

   • On the first page of your write-up, please list students who helped you or whom you helped on the homework. (Note that sending each other code is not allowed.)

2. Submit all the code needed to reproduce your results to the Gradescope assignment entitled "HW4 Code". You must submit your code twice: once in your PDF write-up (above) so the readers can easily read it, and again in compilable/interpretable form so the readers can easily run it. Do NOT include any data files we provided. Please include a short file named README listing your name, student ID, and instructions on how to reproduce your results. Please take care that your code doesn't take up inordinate amounts of time or memory.

For staff use only

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Total
/ 6	/ 6	/ 8	/ 10	/ 10	/ 7	/ 13	/ 10	/ 30	/ 100

Honor Code

Declare and sign the following statement:

"I certify that all solutions in this document are entirely my own and that I have not looked at anyone else's solution. I have given credit to all external sources I consulted."

Signature: I certify that all solutions in this document are entirely my own and that I have not looked at anyone else's solution. I have given credit to all external sources I consulted.

We welcome group discussions, but the work you submit should be entirely your own. If you use any information or pictures not from our lectures or readings, make sure to say where they came from. Please note that breaking academic rules can lead to severe penalties.

(a) Did you receive any help whatsoever from anyone in solving this assignment? If your answer is 'yes', give full details (e.g. "Junho explained to me what is asked in Q2-a")

No.

(b) Did you give any help whatsoever to anyone in solving this assignment? If your answer is 'yes', give full details (e.g. "I pointed Josh to Ch. 2.3 since he didn't know how to proceed with Q2")

No.

(c) Did you find or come across code that implements any part of this assignment? If your answer is 'yes', give full details (book & page, URL & location within the page, etc.).

I reviewed Lab session in Chapter 7 and 8, before I jumped into this assignment.

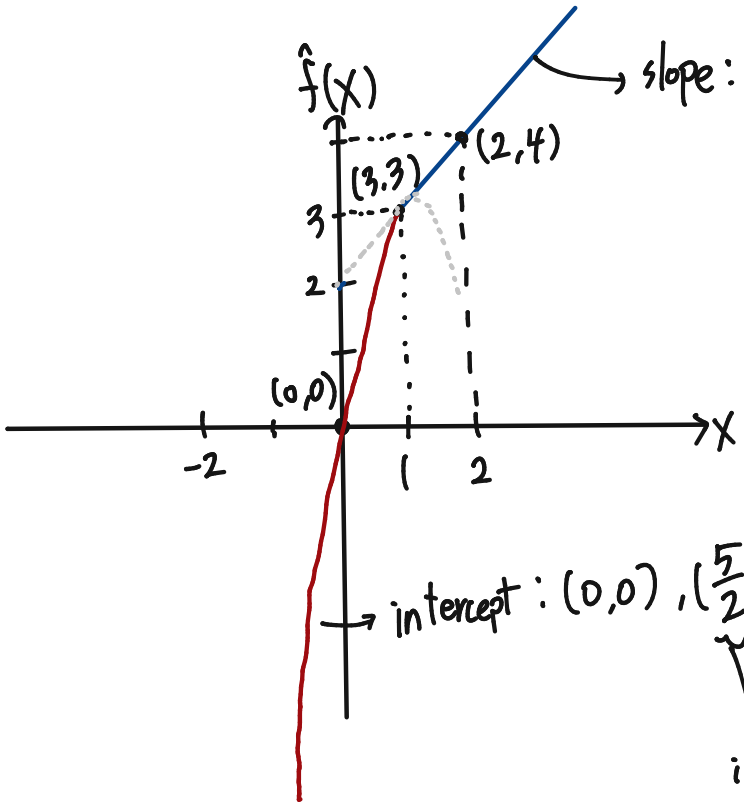
Q1. Basis function [6 pts]

Suppose we fit a curve with basis functions $b_1(X) = X, b_2(X) = (X-1)^2 I(X \leq 1)$. (Note that $I(X \leq 1)$ equals 1 for $X \leq 1$ and 0 otherwise.) We fit the linear regression model $Y = \beta_0 + \beta_1 b_1(X) + \beta_2 b_2(X) + \epsilon$, and obtain coefficient estimates $\hat{\beta}_0 = 2, \hat{\beta}_1 = 1, \hat{\beta}_2 = -2$. Sketch the estimated curve between $X = -3$ and $X = 3$. Note the intercepts, slopes, and other relevant information.

Fitted curve $\hat{f}(X) = \begin{cases} 2 + X - 2(X-1)^2, & \text{for } X \leq 1 \\ 2 + X, & \text{for } X > 1 \end{cases}$

for $X \leq 1$, $\hat{f}(X) = 2 + X - 2(X-1)^2 = -2X^2 + 5X$

$\hat{f}'(X) = -4X + 5$. $\Rightarrow$ when $X = \frac{5}{4}$, $\hat{f}(X)$ has maximum value, but this contradict the assumption $X \leq 1$.



slope : 1, intercept : (0,2), (-2,0) invalid (∵ blue graph is only for $X > 1$)

intercept : (0,0), $(\frac{5}{2}, 0)$, critical point : $(\frac{5}{4}, \frac{25}{8})$ invalid (∵ red graph is only for $X < 1$)

Q2. Analyze splines [6 pts]

Consider two curves $\hat{g}_1$ and $\hat{g}_2$, defined by

$$\hat{g}_1 = \arg\min_g \left(\sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int \left[g^{(3)}(x) \right]^2 dx \right),$$

$$\hat{g}_2 = \arg\min_g \left(\sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int \left[g^{(4)}(x) \right]^2 dx \right),$$

where $g^{(m)}$ represents the mth derivative of g.

(a) As $\lambda \to \infty$, will $\hat{g}_1$ or $\hat{g}_2$ have the smaller training RSS? [3 pts]

$\hat{g}_1$: parameters corresponding to x of the third order or higher go to zero. As a result, $\hat{g}(x)$ approaches a quadratic form, $\hat{g}(x) = ax^2 + bx + c$, minimizing training RSS.

$\hat{g}_2$: Parameters corresponding to x of the fourth order or higher go to zero. This makes $\hat{g}(x)$ cubic form, for example, $\hat{g}(x) = ax^3 + bx^2 + cx + d$, which minimizes training RSS.

∴ Since $\hat{g}(x)$ of $\hat{g}_2$ case is more flexible than that of $\hat{g}_1$ case, $\hat{g}_2$ will get smaller training RSS.

(b) As $\lambda \to \infty$, will $\hat{g}_1$ or $\hat{g}_2$ have the smaller test RSS? [3 pts]

Due to the fact that there are no information about the true relationship between X and Y, I cannot determine which would have the smaller test RSS.

Q3. Sketch splines [8 pts]

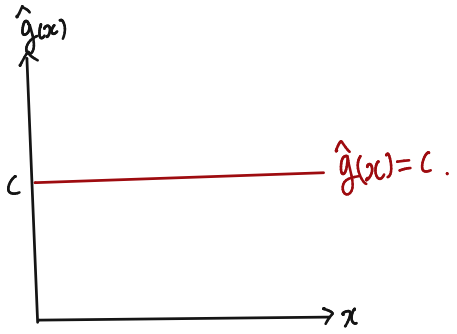
Suppose that a curve $\hat{g}$ is computed to smoothly fit a set of n points using the following formula:

$$\hat{g} = \arg\min_{g} \left(\sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int \left[g^{(m)}(x) \right]^2 dx \right),$$

where $g^{(m)}$ represents the mth derivative of g (and $g^{(0)} = g$). Provide example sketches of $\hat{g}$ in each of the following scenarios.

(a) $\lambda = \infty, m = 1$. [4 pts]

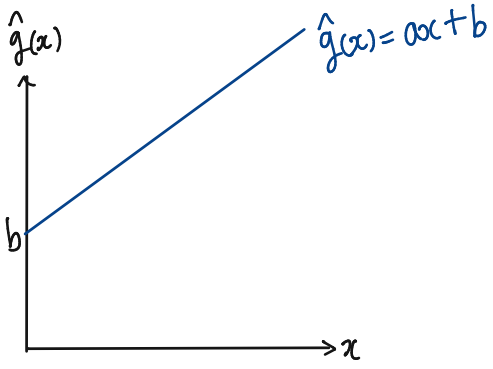
Similar to the previous problem. This condition would make parameters corresponding to x of the first order or higher go to zero. Therefore $\hat{g}(x)$ would be constant function, let's say $\hat{g}(x) = c$.



(b) $\lambda = \infty, m = 2$. [4 pts]

Parameters corresponding to x of second order or higher go to zero.

$\hat{g}(x)$ would be linear, for example, $\hat{g}(x) = ax + b$.



Q4. Drawing a tree and a predictor space [10 pts]

This question relates to the plots below.

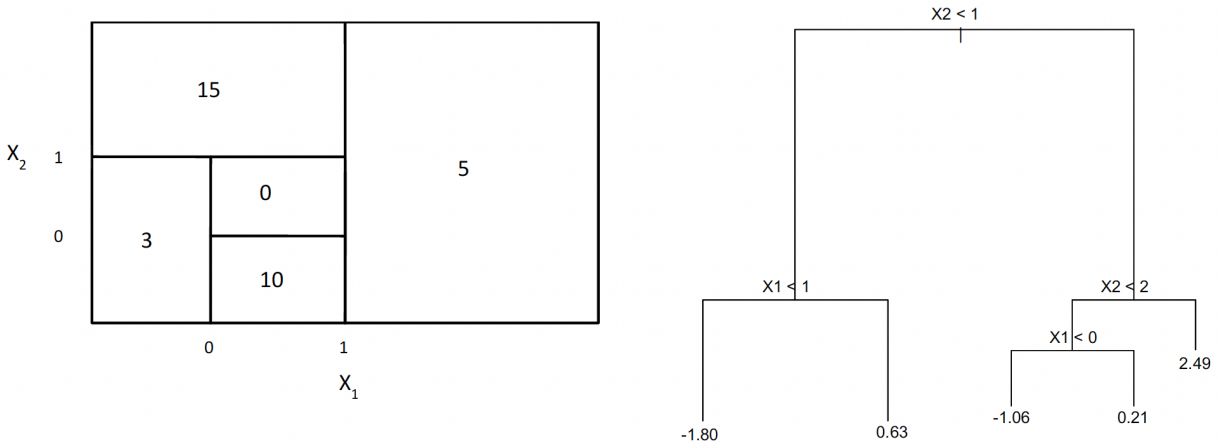
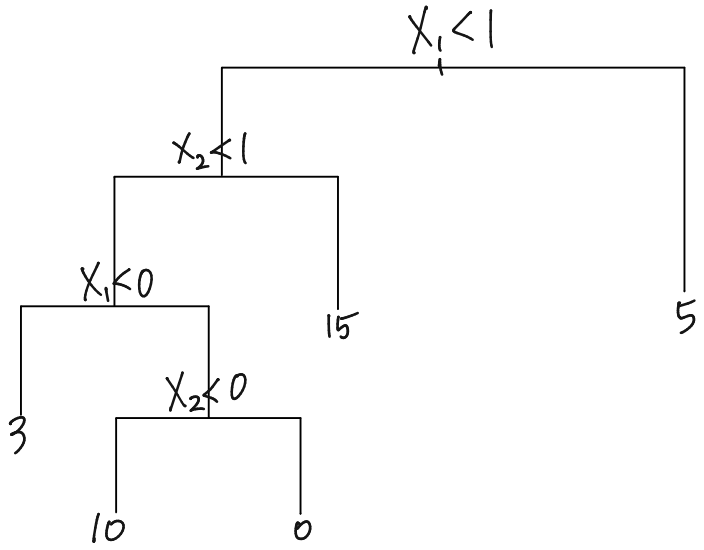
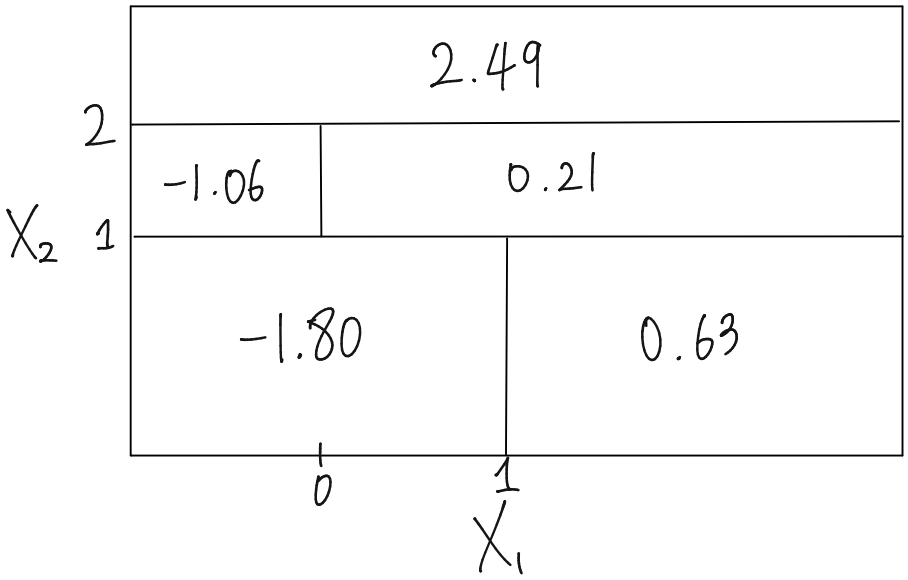


Figure 1: Left: A partition of the predictor space corresponding to problem (a). Right: A tree corresponding to problem (b).

(a) Sketch the tree corresponding to the partition of the predictor space illustrated in the left-hand panel of the Figure 1. The numbers inside the boxes indicate the mean of Y within each region. [5 pts]



(b) Create a diagram similar to the left-hand panel of the Figure 1, using the tree illustrated in the right-hand panel of the same figure. You should divide up the predictor space into the correct regions, and indicate the mean for each region. [5 pts]



Q5. Finding the best fit - Polynomial and Step function [10 pts]

In this exercise, you will further analyze the Wage data set considered throughout this chapter.

(a) Perform polynomial regression to predict wage using age. Use cross-validation to select the optimal degree d for the polynomial. What degree was chosen, and how does this compare to the results of hypothesis testing using ANOVA? Make a plot of the resulting polynomial fit to the data. [6 pts]

```
#1. ANOVA

poly_M_list=[]

for d in range(1,10):
    poly_age = MS([poly('age', degree=d)]).fit(Wage)
    M = sm.OLS(y, poly_age.transform(Wage)).fit()
    poly_M_list.append(M)

anova_list=anova_lm(* poly_M_list)

print(anova_list)
```

[72] ✓ 0.0s Python

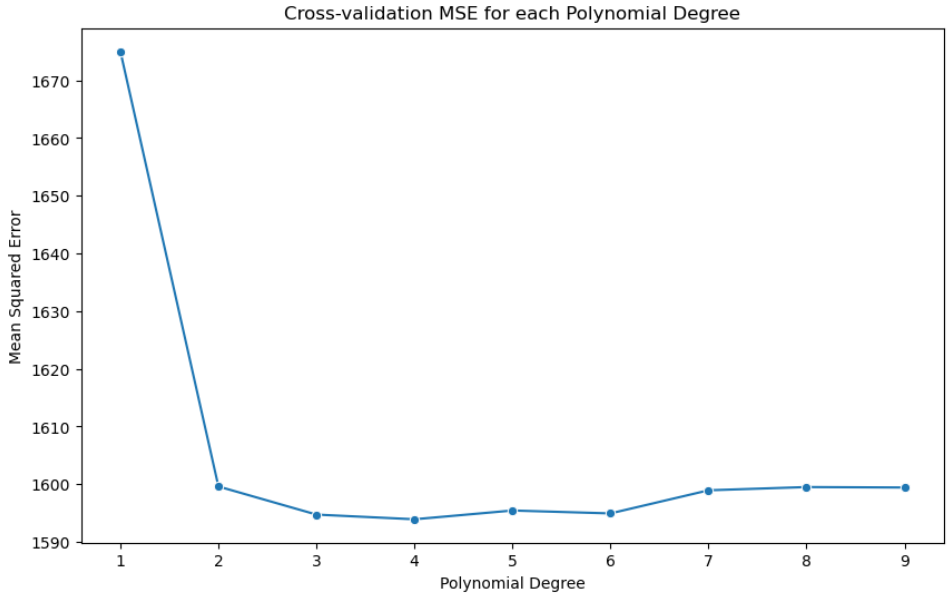
```
   df_resid           ssr  df_diff        ss_diff           F        Pr(>F)
0    2998.0  5.022216e+06      0.0            NaN         NaN           NaN
1    2997.0  4.793430e+06      1.0  228786.010128  143.811827  2.128079e-32
2    2996.0  4.777674e+06      1.0   15755.693664    9.903818  1.665559e-03
3    2995.0  4.771604e+06      1.0    6070.152124    3.815616  5.086990e-02
4    2994.0  4.770322e+06      1.0    1282.563017    0.806202  3.693177e-01
5    2993.0  4.766389e+06      1.0    3932.257665    2.471765  1.160134e-01
6    2992.0  4.763834e+06      1.0    2555.281281    1.606216  2.051232e-01
7    2991.0  4.763707e+06      1.0     126.668985    0.079622  7.778293e-01
8    2990.0  4.756703e+06      1.0    7004.317139    4.402820  3.596326e-02
```

$Pr(> F)$ is the p-value for the F-statistic, so a low value (typically below 0.05) indicates that the new model is statistically significantly better than the previous model.
Moving from a linear to a quadratic model significantly improves fit (very low $Pr(> F) \approx 2.13e-32$). The cubic model is slightly better than the quadratic ($Pr(> F) \approx 1.67e-03$), but higher degrees(where $d \geq 4$) do not provide statistically significant improvements, with the 4th degree and above not considered significant.
Therefore, using ANOVA, the optimal degree is 3, the cubic model.

```python
#2. Cross-validation
degrees = list(range(1, 10))
mse_scores = []

for degree in degrees:
    poly_features = PolynomialFeatures(degree=degree)
    X_poly = poly_features.fit_transform(age.values.reshape(-1, 1))
    model = LinearRegression()

    #Compute MSE after cross-validation
    score = -np.mean(cross_val_score(model, X_poly, y, scoring='neg_mean_squared_error', cv=5))
    mse_scores.append(score)

    print(f"Degree {degree}: Cross-validation MSE = {score:.4f}") #Print the cross-validation scores for degree 1~9

#Finding the optimal degree
optimal_index = np.argmin(mse_scores)
best_degree = degrees[optimal_index]
print(f"\nOptimal polynomial degree: {best_degree}")


scores_df = pd.DataFrame({'Degree': degrees, 'MSE': mse_scores})
plt.figure(figsize=(10, 6))
sns.lineplot(x='Degree', y='MSE', data=scores_df, marker='o')
plt.title('Cross-validation MSE for each Polynomial Degree')
plt.xlabel('Polynomial Degree')
plt.ylabel('Mean Squared Error')
plt.xticks(degrees)
plt.show()
```

[75] ✓ 0.1s Python

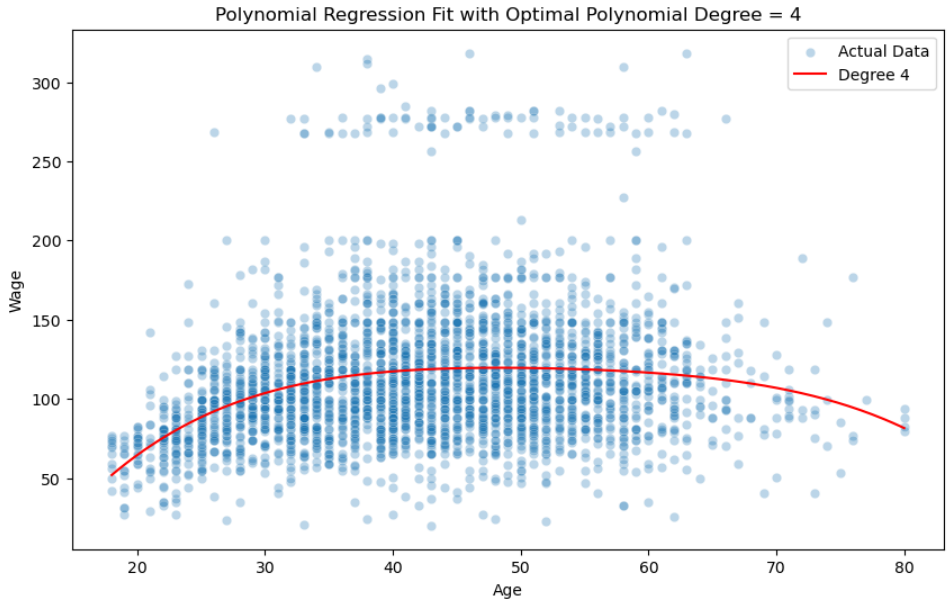
```
Degree 1: Cross-validation MSE = 1675.0142
Degree 2: Cross-validation MSE = 1599.5982
Degree 3: Cross-validation MSE = 1594.7262
Degree 4: Cross-validation MSE = 1593.9141
Degree 5: Cross-validation MSE = 1595.4331
Degree 6: Cross-validation MSE = 1594.9302
Degree 7: Cross-validation MSE = 1598.9376
Degree 8: Cross-validation MSE = 1599.4999
Degree 9: Cross-validation MSE = 1599.4215

Optimal polynomial degree: 4
```



Therefore, using cross-validation, the optimal polynomial degree is 4. This is different from the previous outcome of ANOVA.

```python
#Using the optimal degree=4
poly_features = PolynomialFeatures(degree=best_degree)
X_poly = poly_features.fit_transform(age.values.reshape(-1, 1))
best_poly_model = LinearRegression()
best_poly_model.fit(X_poly, y)

#Generate a range of features for prediction
age_range = np.linspace(age.min(), age.max(), 200).reshape(-1, 1)
age_range_poly = poly_features.transform(age_range)
predictions = best_poly_model.predict(age_range_poly)

#Plot polynomial regression
plt.figure(figsize=(10, 6))
sns.scatterplot(x=age, y=y, alpha=0.3, label='Actual Data')
sns.lineplot(x=age_range.flatten(), y=predictions, color='red', label=f'Degree {best_degree}')
plt.xlabel('Age')
plt.ylabel('Wage')
plt.title('Polynomial Regression Fit with Optimal Polynomial Degree = 4')
plt.legend()
plt.show()
```

[76] ✓ 0.2s Python

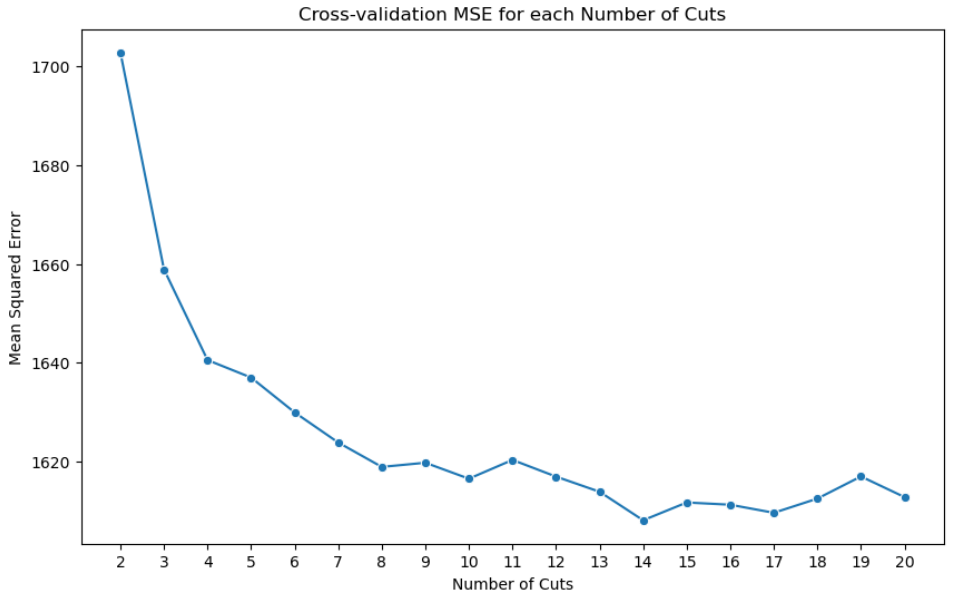


I made a graph above which is the plot of the resulting polynomial fit to the data.

(b) Fit a step function to predict wage using age, and perform cross-validation to choose the optimal number of cuts. Make a plot of the fit obtained. [4 pts]

```python
cuts_list = list(range(2, 21))
mse_scores = []
for cuts in cuts_list:
    cut_age = pd.qcut(age, q=cuts)
    X_cut = pd.get_dummies(cut_age)
    model = LinearRegression()

    #Compute MSE after cross-validation
    score = -np.mean(cross_val_score(model, X_cut, y, scoring='neg_mean_squared_error', cv=5))
    mse_scores.append(score)

    #Print the cross-validation scores for cuts 2~20
    print(f"Cuts {cuts}: Cross-validation MSE = {score:.4f}")

optimal_index = np.argmin(mse_scores)
best_cuts = cuts_list[optimal_index]
print(f"\nOptimal number of cuts: {best_cuts}")

#Plot the cross-validation mse
cuts_df = pd.DataFrame({'Cuts': cuts_list, 'MSE': mse_scores})
plt.figure(figsize=(10, 6))
sns.lineplot(x='Cuts', y='MSE', data=cuts_df, marker='o')
plt.title('Cross-validation MSE for each Number of Cuts')
plt.xlabel('Number of Cuts')
plt.ylabel('Mean Squared Error')
plt.xticks(cuts_list)
plt.show()
```

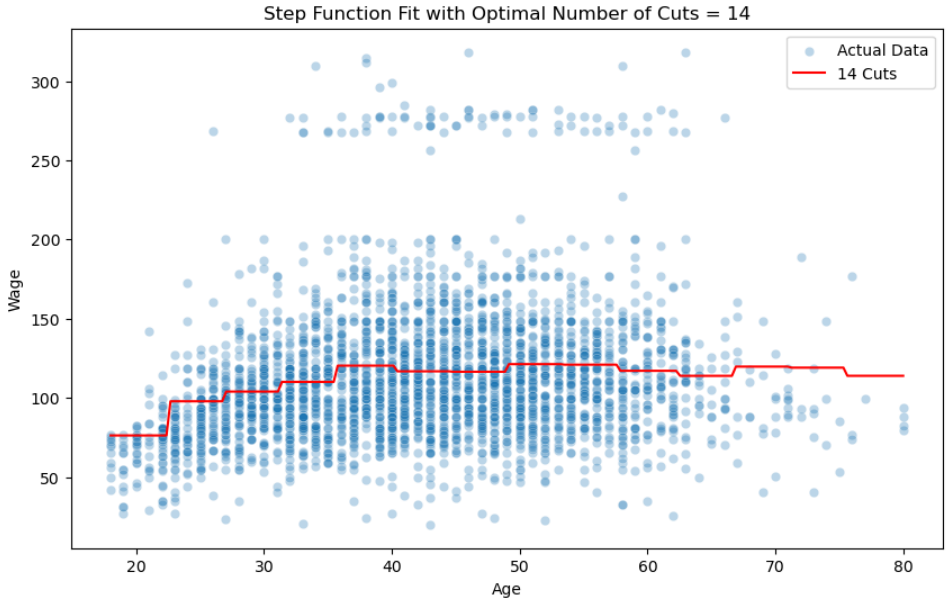
[51] Python



I can increase the number of cuts until I reach the unique number of values in the age variable, but this equates to having only one data point for each interval, which makes it impossible to make meaningful generalizations in practice.
Therefore, I just set the range of cuts from 2 to 20, and as a result of cross-validation, ‘cuts=14‘ had the lowest mse.

```python
#Using the optimal cuts=14
cut_age = pd.qcut(age, q=best_cuts)
X_cut = pd.get_dummies(cut_age)
best_step_model = sm.OLS(y, X_cut).fit()

#Generate a range of features for prediction
age_range = np.linspace(age.min(), age.max(), 200)
cut_age_range = pd.qcut(age_range, q=best_cuts, labels=False, duplicates='drop')
X_cut_range = pd.get_dummies(cut_age_range)
predictions = best_step_model.predict(X_cut_range)

#Plot step function
plt.figure(figsize=(10, 6))
sns.scatterplot(x=age, y=y, alpha=0.3, label='Actual Data')
sns.lineplot(x=age_range, y=predictions, color='red', label=f'{best_cuts} Cuts')
plt.xlabel('Age')
plt.ylabel('Wage')
plt.title(f'Step Function Fit with Optimal Number of Cuts = {best_cuts}')
plt.legend()
plt.show()
```

[78] ✓ 0.1s Python



The graph above is the step function fit obtained with optimal cuts=14.

Q6. Finding the best fit - Regression Spline [7 pts]

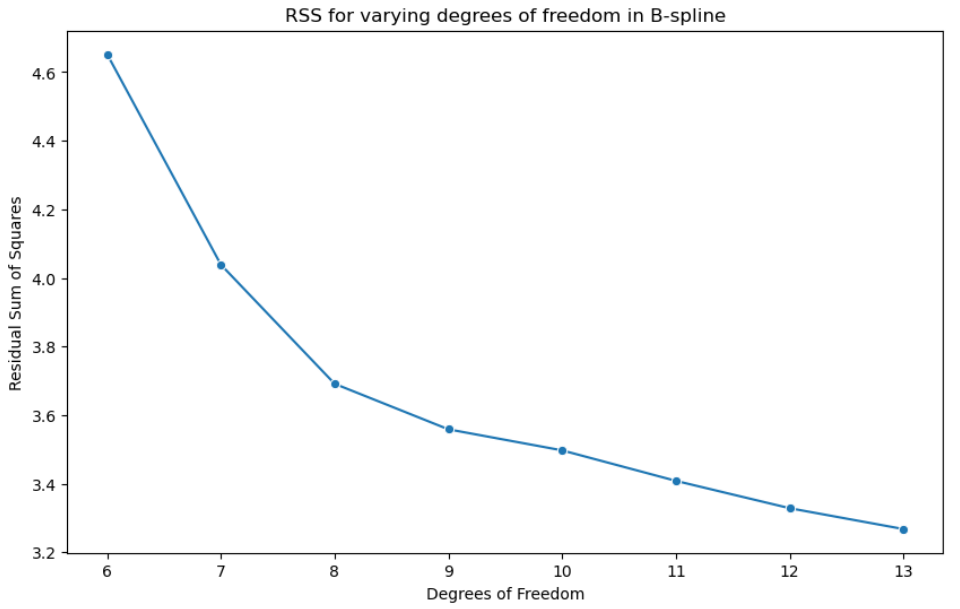
This question uses the variables dis (the weighted mean of distances to five Boston employment centers) and nox (nitrogen oxides concentration in parts per 10 million) from the Boston data. We will treat dis as the predictor and nox as the response.

(a) Use the bs() function from the ISLP.models to fit a regression spline to predict nox using dis. Report the resulting RSS by changing a range of degrees of freedom. Describe the results obtained. [4 pts]

```
y = Boston['nox']
dis = Boston['dis'].values.reshape(-1, 1)

degrees_of_freedom = list(range(6,14))
rss_list=[]

for degree in degrees_of_freedom:
    #Generate new B-spline features for the 'dis' feature with varying degrees of freedom
    bspl = BSpline(df=degree).fit(dis)  #we should fit BSpline on the 'dis' column only
    knots_list = bspl.internal_knots_
    bs_dis = bspl.transform(dis)  #Transform 'dis' using the fitted BSpline

    M = sm.OLS(y, bs_dis).fit()

    predictions = M.predict(bs_dis)
    residuals = y - predictions
    rss = np.sum(residuals**2)
    rss_list.append(rss)
    print(degree, rss)  #Print RSS for each degree of freedom

plt.figure(figsize=(10, 6))
sns.lineplot(x=degrees_of_freedom, y=rss_list, marker='o')
plt.title('RSS for varying degrees of freedom in B-spline')
plt.xlabel('Degrees of Freedom')
plt.ylabel('Residual Sum of Squares')
plt.xticks(degrees_of_freedom)
plt.show()
```

[92] ✓ 0.0s Python

```
6 4.651670452174841
7 4.038403724692655
8 3.691307386881299
9 3.558645592424596
10 3.4972805032423318
11 3.408442220337952
12 3.3287533881628653
13 3.2679567729069436
```



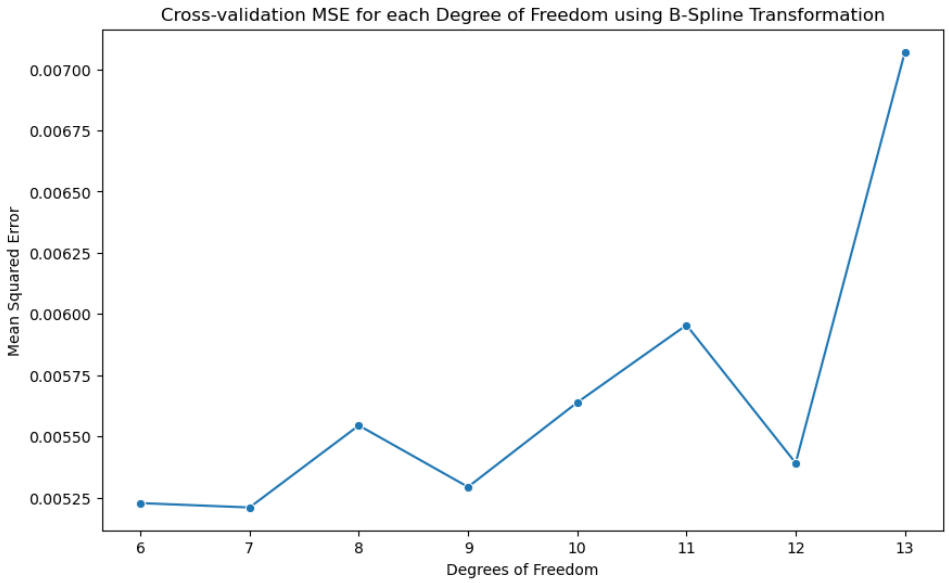
By the graph above, I can observe that the RSS decreases when the degrees of freedom increase.

(b) Perform cross-validation or another approach in order to select the best degrees of freedom for a regression spline on this data, and explain your results. [3 pts]

```python
#Cross-validation
degrees_of_freedom = list(range(6, 14))
mse_scores = []

for degree in degrees_of_freedom:
    #Generate new B-spline features for the 'dis' feature with varying degrees of freedom
    bspl = BSpline(df=degree).fit(dis) #Fit BSpline on the 'dis' column only
    knots_list = bspl.internal_knots_
    bs_dis = bspl.transform(dis) #Transform 'dis' using the fitted BSpline
    model = LinearRegression()

    score = -np.mean(cross_val_score(model, bs_dis, y, scoring='neg_mean_squared_error', cv=5))
    mse_scores.append(score)

    #Print the cross-validation scores for degrees of freedom 6~13
    print(f"Degrees of Freedom {degree}: Cross-validation MSE = {score:.4f}")

#Finding the optimal degrees of freedom
optimal_index = np.argmin(mse_scores)
best_df = degrees_of_freedom[optimal_index]
print(f"\nOptimal degrees of freedom: {best_df}")
print(f"Knots are {BSpline(df=best_df).fit(dis).internal_knots_}")

scores_df = pd.DataFrame({'Degrees of Freedom': degrees_of_freedom, 'MSE': mse_scores})
plt.figure(figsize=(10, 6))
sns.lineplot(x='Degrees of Freedom', y='MSE', data=scores_df, marker='o')
plt.title('Cross-validation MSE for each Degree of Freedom using B-Spline Transformation')
plt.xlabel('Degrees of Freedom')
plt.ylabel('Mean Squared Error')
plt.xticks(degrees_of_freedom)
plt.show()
```

[93] ✓ 0.1s Python

```
Degrees of Freedom 6: Cross-validation MSE = 0.0052
Degrees of Freedom 7: Cross-validation MSE = 0.0052
Degrees of Freedom 8: Cross-validation MSE = 0.0055
Degrees of Freedom 9: Cross-validation MSE = 0.0053
Degrees of Freedom 10: Cross-validation MSE = 0.0056
Degrees of Freedom 11: Cross-validation MSE = 0.0060
Degrees of Freedom 12: Cross-validation MSE = 0.0054
Degrees of Freedom 13: Cross-validation MSE = 0.0071

Optimal degrees of freedom: 7
Knots are [1.9512 2.6403 3.875  5.615 ]
```



Through the graph above, I found that if the degree of freedom is too high, it may be seen that the MSE may increase due to overfitting. And the optimal degrees of freedom is 7, which means that there are 4 knots : ‘[1.9512, 2.6403, 3.875, 5.615]‘

Q7. Regression trees [13 pts]

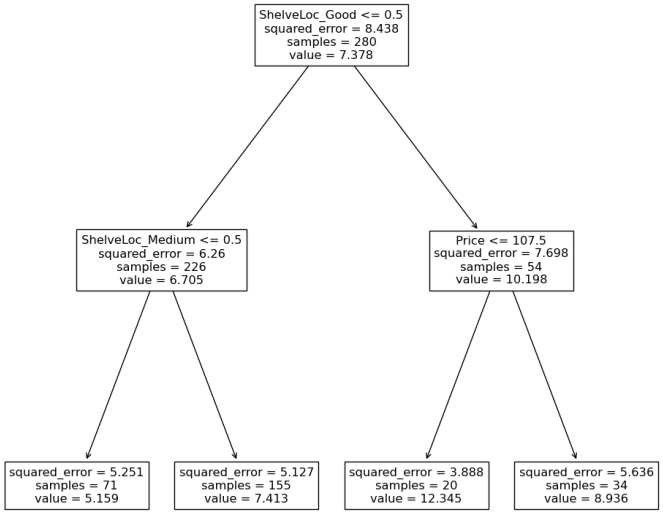
In the lab, a classification tree was applied to the Carseats data set after converting Sales into a qualitative response variable. Now we will seek to predict Sales using regression trees and related approaches, treating the response as a quantitative variable.

(a) Split the data into a training and a test set, and fit a regression tree to the training set. Plot the tree, and interpret the results. What test MSE do you obtain? [3 pts]

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

#I found that there are object types in data set, so we need to add dummy variables.
Carseats = pd.get_dummies(Carseats)

#Split the data set into train set and test set.
train_set, test_set = train_test_split(Carseats, test_size=0.3, random_state=0)

X_train=train_set.drop(columns=['Sales'])
y_train=train_set['Sales']

X_test=test_set.drop(columns=['Sales'])
y_test=test_set['Sales']

#I used DicisionTreeRegressor.
#And I briefly set max_depth to two, I will compute optimal level of tree complexity in next question.
reg=DTR(max_depth=2, random_state=0)
reg.fit(X_train, y_train)

#print test MSE
y_pred = reg.predict(X_test)
test_mse = mean_squared_error(y_test, y_pred)
print(f"Test MSE: {test_mse:.4f}")

ax=subplots(figsize=(12,12))[1]
plot_tree(reg, feature_names=list(X_train.columns), ax=ax)
```

[106] ✓ 0.1s Python

```
Test MSE: 5.0176

[Text(0.5, 0.8333333333333334, 'ShelveLoc_Good <= 0.5\nsquared_error = 8.438\nsamples = 280\nvalue = 7.378'),
 Text(0.25, 0.5, 'ShelveLoc_Medium <= 0.5\nsquared_error = 6.26\nsamples = 226\nvalue = 6.705'),
 Text(0.125, 0.16666666666666666, 'squared_error = 5.251\nsamples = 71\nvalue = 5.159'),
 Text(0.375, 0.16666666666666666, 'squared_error = 5.127\nsamples = 155\nvalue = 7.413'),
 Text(0.75, 0.5, 'Price <= 107.5\nsquared_error = 7.698\nsamples = 54\nvalue = 10.198'),
 Text(0.625, 0.16666666666666666, 'squared_error = 3.888\nsamples = 20\nvalue = 12.345'),
 Text(0.875, 0.16666666666666666, 'squared_error = 5.636\nsamples = 34\nvalue = 8.936')]
```



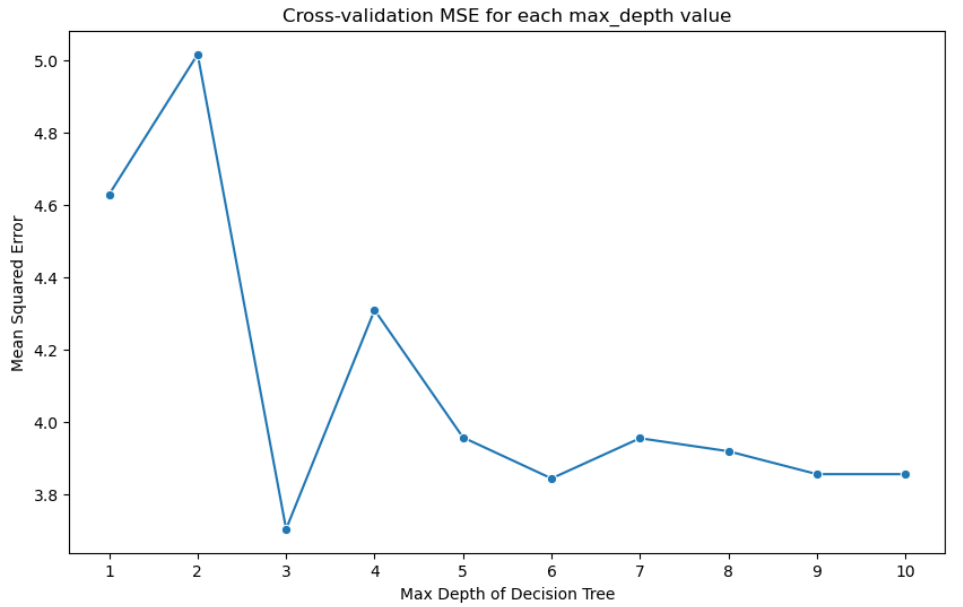
The root node of the tree uses 'ShelveLoc Good' as an important feature for splitting. The leaf node with the most data satisfies 'ShelveLoc Good ≤ 0.5' and 'ShelveLoc Medium > 0.5'.
Since the max depth was set to 2, there is a possibility that the model is underfit. It is necessary to determine the optimal level of tree complexity through cross-validation in the next problem.
As the tree undergoes more splits, each node ends up with fewer data points, making it more specific to the dataset and reducing the variance(as well as squared error) within the node's data.
Finally, the test MSE I obtained is 5.0176.

(b) Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test MSE? [3 pts]

```python
max_depths = list(range(1, 11))
mse_scores = []

for max_depth in max_depths:
    reg=DTR(max_depth=max_depth,random_state=0)

    ccp_path = reg.cost_complexity_pruning_path(X_train, y_train)
    kfold = skm.KFold(5, shuffle=True, random_state=0)
    grid = skm.GridSearchCV(reg, {'ccp_alpha': ccp_path.ccp_alphas}, refit=True, cv=kfold, scoring='neg_mean_squared_error')
    G = grid.fit(X_train, y_train)

    y_pred = G.predict(X_test)
    test_mse = mean_squared_error(y_test, y_pred)
    mse_scores.append(test_mse)
    print(f"Max_depth {max_depth}: Test MSE: {test_mse:.4f}") #Print the cross-validation Test MSE for max_depth 1~10

#Finding the optimal max_depth
optimal_index = np.argmin(mse_scores)
best_depth = max_depths[optimal_index]
print(f"\nOptimal max_depth: {best_depth}")

scores_df = pd.DataFrame({'Max_depth': max_depths, 'MSE': mse_scores})
plt.figure(figsize=(10, 6))
sns.lineplot(x='Max_depth', y='MSE', data=scores_df, marker='o')
plt.title('Cross-validation MSE for each max_depth value')
plt.xlabel('Max Depth of Decision Tree')
plt.ylabel('Mean Squared Error')
plt.xticks(max_depths)
plt.show()
```

[99] ✓ 8.8s Python

```
Max_depth 1: Test MSE: 4.6303
Max_depth 2: Test MSE: 5.0176
Max_depth 3: Test MSE: 3.7039
Max_depth 4: Test MSE: 4.3099
Max_depth 5: Test MSE: 3.9573
Max_depth 6: Test MSE: 3.8438
Max_depth 7: Test MSE: 3.9552
Max_depth 8: Test MSE: 3.9192
Max_depth 9: Test MSE: 3.8558
Max_depth 10: Test MSE: 3.8558

Optimal max_depth: 3
```



Pruning the decision tree to a maximum depth of 3 optimizes its performance.
I found that when the the 'max depth' of decision tree is 3, we can have the lowest MSE.
For 'max depth $>$ 3', increasing tree complexity results in higher MSE, indicating overfitting.
Therefore, tree pruning improves the test MSE for this dataset.

(c) Use the bagging approach in order to analyze this data. What test MSE do you obtain? Use the feature_importance_ values to determine which variables are most important. [3 pts]

```
#BaggingRegressor
#Bagging is a special case of a Randomforest with m=p.
#And also, Bagging is the method that use all p predictors, so I made max_features=X_train.shape[1].

bgr = RF(max_features=X_train.shape[1],random_state=0)
bgr.fit(X_train, y_train)
y_pred = bgr.predict(X_test)
test_mse = mean_squared_error(y_test, y_pred)
print(f"Test MSE: {test_mse:.4f}")
```

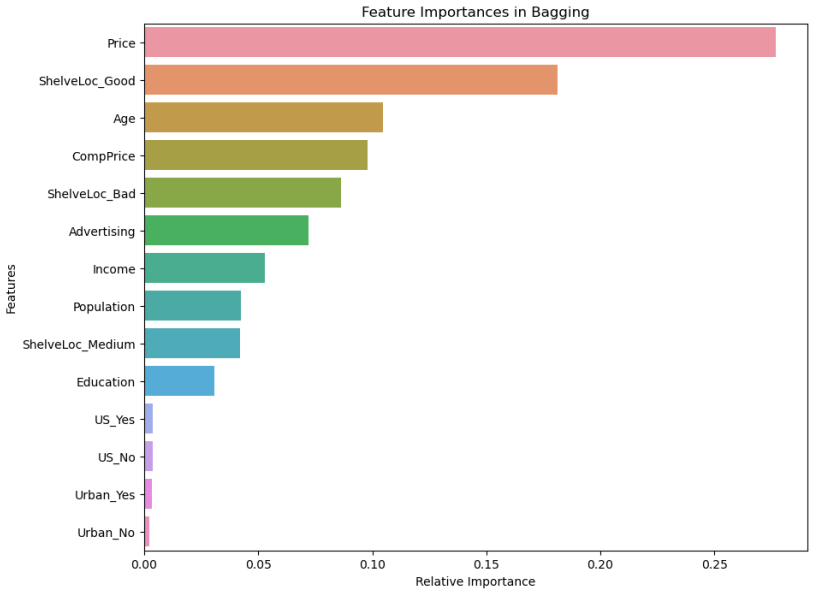
[100] ✓ 0.1s Python

```
Test MSE: 2.1006
```

```
#I created a dataframe of feature importances by using feature_importances_
feature_imp = pd.DataFrame({'Feature': X_train.columns, 'Importance': bgr.feature_importances_})
#Sortting by importance in descending order
feature_imp = feature_imp.sort_values('Importance', ascending=False)
print(feature_imp)

#I plotted a bar chart for visualization
plt.figure(figsize=(10, 8))
sns.barplot(x='Importance', y='Feature', data=feature_imp)
plt.title('Feature Importances in Bagging')
plt.xlabel('Relative Importance')
plt.ylabel('Features')
plt.show()
```

[101] ✓ 0.1s Python

```
          Feature  Importance
4           Price    0.276948
8   ShelveLoc_Good    0.181296
5             Age    0.104809
0       CompPrice    0.097882
7    ShelveLoc_Bad    0.086165
2     Advertising    0.072039
1          Income    0.053028
3      Population    0.042258
9  ShelveLoc_Medium    0.041852
6       Education    0.030831
13        US_Yes    0.003634
12         US_No    0.003626
11      Urban_Yes    0.003259
10       Urban_No    0.002373
```



The test MSE I obtained is 2.1006.
Through the graph above, I found that 'Price' is the most important varibles among all features.

(d) Use random forests to analyze this data. What test MSE do you obtain? Use the feature_importance_ function to determine which variables are most important. Describe the effect of m, the number of variables considered at each split, on the error rate obtained. [4 pts]

```python
#Randomforest
#In randomforest, we use the m=p^0.5. Therefore, we should set max_features=int(X_train.shape[1]**0.5).
#But I realized that the RF automatically set max_features properly if I set max_features='sqrt'.

rf = RF(max_features='sqrt', random_state=0)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
test_mse = mean_squared_error(y_test, y_pred)
print(f"Test MSE: {test_mse:.4f}")
```

[102] ✓ 0.0s Python

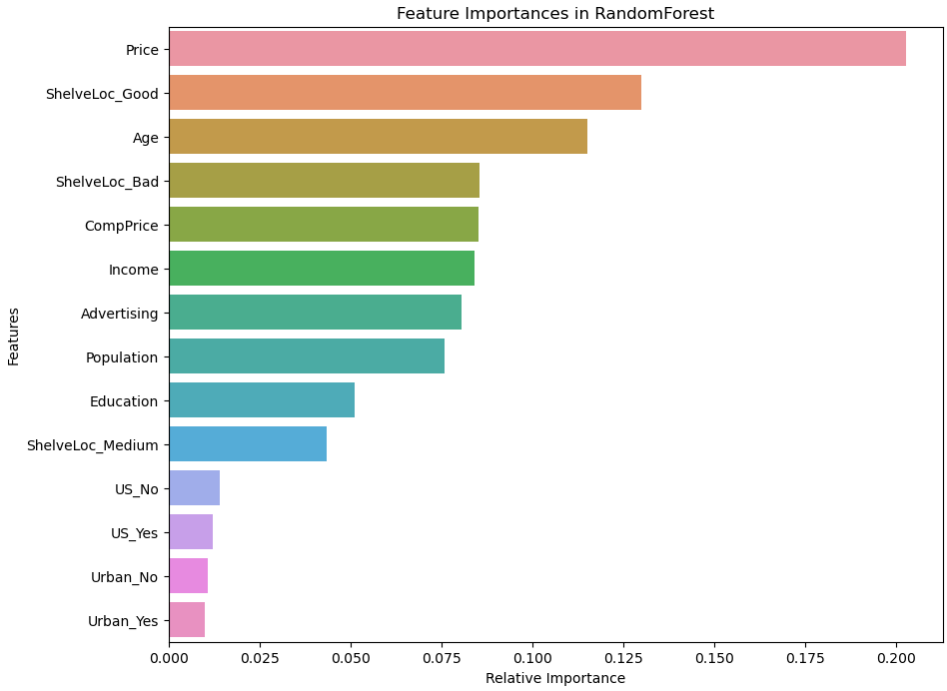
```
Test MSE: 2.4892
```

```python
#I created a dataframe of feature importances by using feature_importances_
feature_imp = pd.DataFrame({'Feature': X_train.columns, 'Importance': rf.feature_importances_})
#Sortting by importance in descending order
feature_imp = feature_imp.sort_values('Importance', ascending=False)
print(feature_imp)

#I plotted a bar chart for visualization
plt.figure(figsize=(10, 8))
sns.barplot(x='Importance', y='Feature', data=feature_imp)
plt.title('Feature Importances in RandomForest')
plt.xlabel('Relative Importance')
plt.ylabel('Features')
plt.show()
```

[103] ✓ 0.1s Python

```
            Feature  Importance
4             Price    0.202796
8    ShelveLoc_Good    0.130043
5               Age    0.114988
7    ShelveLoc_Bad    0.085509
0         CompPrice    0.085291
1            Income    0.084190
2       Advertising    0.080447
3        Population    0.075802
6         Education    0.051163
9  ShelveLoc_Medium    0.043449
12            US_No    0.013951
13           US_Yes    0.011952
10         Urban_No    0.010576
11        Urban_Yes    0.009842
```

The test MSE I obtained is 2.4892.



Just like problem (c), Price is the most important feature.
If we see the graph above, 'Price' is the most important varibles among all features.

Finally, let's describe the effect of m, the number of variables considered at each split, on the error rate obtained.
In comparison to case (c) where 'm=p' led to 'max features' being set to 14, this problem's setting of 'm=sqrt(p)' results in a 'max features' value of 3. This is a relatively small number of features to consider at each split and could cause the model to miss capturing significant patterns in the data.
As a result, the test MSE in case (d) turned out to be higher than that in case (c). And also, this had an impact on the calculated feature importance. For instance, the importance of 'Price' is seen to diminish in (d), which implies that a smaller 'max features' value can lead the model to overlook essential patterns.

Q8. Boosted trees and its competitors [10 pts]

We now use boosting to predict Salary in the Hitters data set.

(a) Remove the observations for whom the salary information is unknown, and then log-transform the salaries. Next, create a training set consisting of the first 200 observations, and a test set consisting of the remaining observations. Perform boosting on the training set with 1,000 trees for a range of values of the shrinkage parameter λ. Produce two plots: one with different shrinkage values on the x-axis and the corresponding training set MSE on the y-axis, another with different shrinkage values on the x-axis and the corresponding test set MSE on the y-axis. [5 pts]

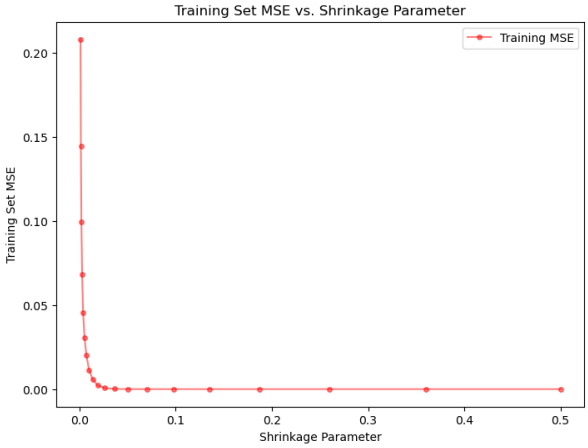
```python
Hitter=load_data("Hitters")

#I removed 59 NA observations.
Hitter = Hitter.dropna(subset=['Salary'])

#Log-transform the 'Salary'
Hitter['Salary'] = np.log(Hitter['Salary'])

#There are several category type predictors. So I needed to deal with them.
Hitter = pd.get_dummies(Hitter)

#Training set with the first 200 observations, and Test set with remaining observations
train_set, test_set = Hitter.iloc[:200], Hitter.iloc[200:]

X_train=train_set.drop(columns=['Salary'])
y_train=train_set['Salary']

X_test=test_set.drop(columns=['Salary'])
y_test=test_set['Salary']
```

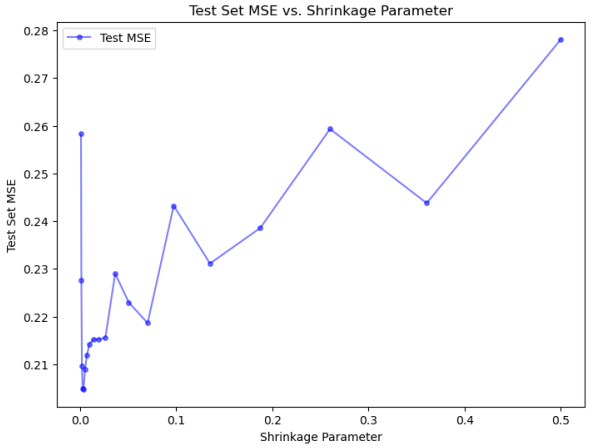
[57] ✓ 0.0s Python

```python
#Shrinkage parameter is learning rate in GradientBoostingRegressor()

# Generate 20 logarithmically spaced values for learning rates between 0.001 and 0.5
learning_rates = np.logspace(np.log10(0.001), np.log10(0.5), num=20)
train_mse, test_mse = [], []

for lr in learning_rates: #lr is the learning rate, the shrinkage parameter lamda.
    #n_estimators indicates the number of trees, and I just breifly set max_depth to 3.
    gbr = GBR(n_estimators=1000, learning_rate=lr, max_depth=3, random_state=0)
    gbr.fit(X_train, y_train)

    #Training set MSE
    train_pred = gbr.predict(X_train)
    train_mse.append(mean_squared_error(y_train, train_pred))

    #Test set MSE
    test_pred = gbr.predict(X_test)
    test_mse.append(mean_squared_error(y_test, test_pred))

#Find the index of the minimum MSE score for the training and test sets
optimal_train_index, optimal_test_index = np.argmin(train_mse), np.argmin(test_mse)
#Find the optimal learning rates based on the minimum MSE scores
optimal_train_lr, optimal_test_lr = learning_rates[optimal_train_index], learning_rates[optimal_test_index]
#Output the minimum MSE and corresponding learning rates
print(f"Training set : when shrinkage parameter = {optimal_train_lr}, minimum MSE = {train_mse[optimal_train_index]:.4f}")
print(f"Test set : when shrinkage parameter = {optimal_test_lr}, minimum MSE = {test_mse[optimal_test_index]:.4f}")

#Plot MSE
fig, axes = plt.subplots(ncols=2, figsize=(18, 6))
axes[0].plot(learning_rates, train_mse, label='Training MSE', marker='o', color='red', markersize=4, alpha=0.5)
axes[0].set_xlabel('Shrinkage Parameter')
axes[0].set_ylabel('Training Set MSE')
axes[0].set_title('Training Set MSE vs. Shrinkage Parameter')
axes[0].legend()
axes[1].plot(learning_rates, test_mse, label='Test MSE', marker='o', color='blue', markersize=4, alpha=0.5)
axes[1].set_xlabel('Shrinkage Parameter')
axes[1].set_ylabel('Test Set MSE')
axes[1].set_title('Test Set MSE vs. Shrinkage Parameter')
axes[1].legend()
plt.show()
```

[60] ✓ 11.7s Python

```
Training set : when shrinkage parameter = 0.5, minimum MSE = 0.0000
Test set : when shrinkage parameter = 0.0037000211913891527, minimum MSE = 0.2048
```





Seeing the graph above, I could found that when the Shrinkage Parameter, the learning rate is too high, there might be Overfitting issues when we applying model in test data set.

(b) Compare the test MSE of boosting to the test MSE that results from applying two of the regression approaches seen in Chapter 3 and 6 of the book. [3 pts]

```python
from sklearn.linear_model import LinearRegression, LassoCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

#Linear regression
#I made a pipeline to perform scaling.
linear_pipeline = make_pipeline(StandardScaler(), LinearRegression())
linear_pipeline.fit(X_train, y_train)
linear_test_pred = linear_pipeline.predict(X_test)
linear_test_mse = mean_squared_error(y_test, linear_test_pred)

#Lasso Regression
#I made a pipeline to perform scaling and finding an appropriate alphas through 10-fold cross-validation in LassoCV
lasso_cv_pipeline = make_pipeline(StandardScaler(), LassoCV(alphas=None, cv=5, max_iter=1000))
lasso_cv_pipeline.fit(X_train, y_train)
lasso_cv_test_pred = lasso_cv_pipeline.predict(X_test)
lasso_cv_test_mse = mean_squared_error(y_test, lasso_cv_test_pred)

print(f"Linear Regression Test MSE: {linear_test_mse:.4f}")
print(f"Lasso Regression Test MSE: {lasso_cv_test_mse:.4f}")
```

[61] ✓ 0.0s Python

```
Linear Regression Test MSE: 0.4918
Lasso Regression Test MSE: 0.4561
```

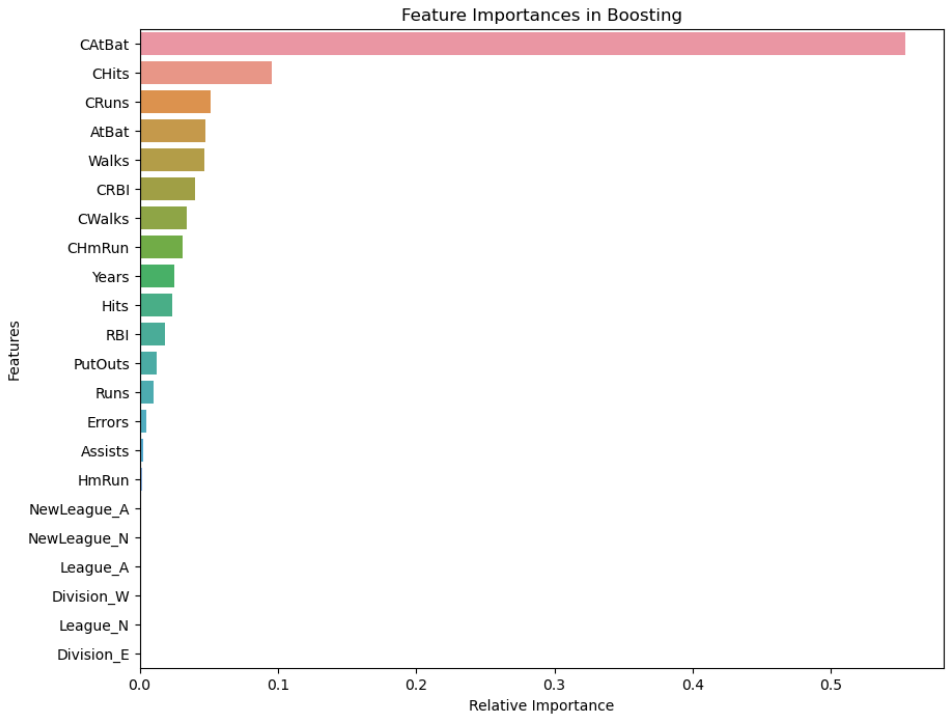
In problem (a), the minimum test MSE of boosting was about '0.2048'. And this is much smaller than the value above in problem (b).
Based on these results, I conclude that boosting performs better than both Linear Regression and Lasso Regression in this scenario.

(c) Which variables appear to be the most important predictors in the boosted model? [2 pts]

```
best_gbr = GBR(n_estimators=1000, learning_rate=optimal_test_lr, max_depth=3, random_state=0)
best_gbr.fit(X_train, y_train)

feature_imp = pd.DataFrame({'Feature': X_train.columns, 'Importance': best_gbr.feature_importances_})
feature_imp = feature_imp.sort_values('Importance', ascending=False)
print(feature_imp)

plt.figure(figsize=(10, 8))
sns.barplot(x='Importance', y='Feature', data=feature_imp)
plt.title('Feature Importances in Boosting')
plt.xlabel('Relative Importance')
plt.ylabel('Features')
plt.show()
```

[67] ✓ 0.7s Python

```
        Feature  Importance
7         CAtBat    0.553599
8          CHits    0.095809
10         CRuns    0.050936
0          AtBat    0.047326
5          Walks    0.046930
11          CRBI    0.040144
12        CWalks    0.034140
9         CHmRun    0.031102
6          Years    0.025038
1           Hits    0.023210
4            RBI    0.018289
13       PutOuts    0.012396
3           Runs    0.009679
15        Errors    0.004991
14       Assists    0.002574
2          HmRun    0.001590
20  NewLeague_A    0.000725
21  NewLeague_N    0.000553
16     League_A    0.000351
19   Division_W    0.000229
17     League_N    0.000211
18   Division_E    0.000180
```



This is similar to the Q7(c).
Through the graph above, I found that 'CAtBat' is the most important variables among all features.

Q9. Mini-competition: Predict flight ticket prices [30 pts]

This marks the final challenge of HW4. Your task is to predict flight ticket prices. Use your student ID, strive to surpass the target score, and submit your results. You're free to use any libraries available - such as scikit-learn, pandas, numpy, statsmodel, ISLP, xgboost, polars and others. After finish your work, capture your public leaderboard score and your code as evidence in your HW4 write-up. Follow these simple rules: Use your student ID, play fair, give it your all, aim high, and, most importantly, enjoy the mini-competition: https://www.kaggle.com/competitions/gist-mldl23f-hw4.

If the above link does not work, please use this invitation link:
https://www.kaggle.com/t/552ac10de4c3495b95d998e79bfd8da4.

Score guideline

- Final points will be determined by the private leaderboard score s and the private target score t.
- Who achieved better the target score $(s \leq 0.1233)$ will receive 30 points.
- Those who didn't beat the target score ($0.1233 < s$) will receive $\max(30(1-s), 0)$ points.
- You're free to use the discussion tab for sharing your thoughts and code snippets. (If you want!)
- However, those who break the honor code will be handled accordingly.
  (Remember collaboration policy & honor code at https://sundong.kim/courses/mldl23f/info)

#	Team	Members	Score	Entries	Last
1	20215169		0.0573	12	15d
2	20214072		0.0596	1	3d
3	20234074		0.0604	14	2d
4	20234075		0.0628	2	5d
5	20234072		0.0638	2	27m
6	20205133		0.0640	1	18d
7	**20195024**		0.0641	5	2m

Your Best Entry!
Your most recent submission scored 0.0641, which is an improvement of your previous score of 0.0651. Great job!

Tweet this

```
train = pd.read_csv('train.csv')
test=pd.read_csv('test.csv')
submission=pd.read_csv('submission.csv')
```
[31] ✓ 0.3s Python

```
train.head()
```
[32] ✓ 0.0s Python

	id	date	airline	ch_code	num_code	dep_time	from	time_taken	stop	arr_time	to	class	price
0	UVYA93	2022-02-11	Indigo	6E	2106	03:00	Delhi	02h 10m	non-stop	05:10	Kolkata	economy	6270
1	1Y6UUB	2022-02-11	AirAsia	I5	764	04:25	Delhi	10h 20m	1-stop	14:45	Bangalore	economy	7423
2	J62B2K	2022-02-11	AirAsia	I5	764	04:25	Delhi	02h 10m	non-stop	06:35	Mumbai	economy	5956
3	HW31QF	2022-02-11	AirAsia	I5	548	04:45	Delhi	02h 25m	non-stop	07:10	Kolkata	economy	6060
4	AI4BTM	2022-02-11	AirAsia	I5	548	04:45	Delhi	14h 20m	1-stop	19:05	Bangalore	economy	21343

```
train.info()
```
[35] ✓ 0.0s Python

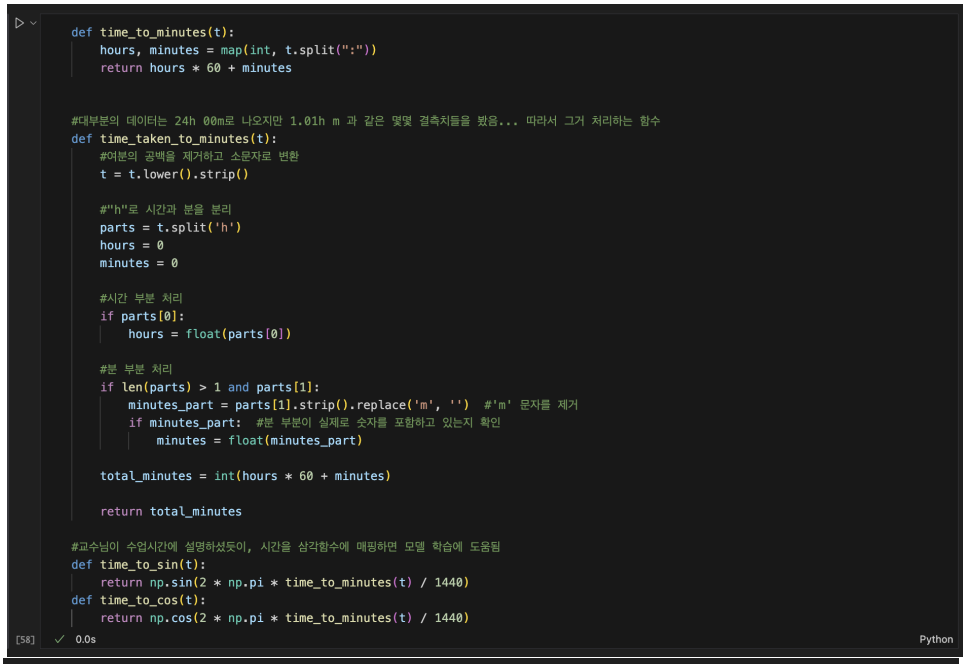
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 243538 entries, 0 to 243537
Data columns (total 13 columns):
 #   Column      Non-Null Count   Dtype 
---  ------      --------------   ----- 
 0   id          243538 non-null  object
 1   date        243538 non-null  object
 2   airline     243538 non-null  object
 3   ch_code     243538 non-null  object
 4   num_code    243538 non-null  int64 
 5   dep_time    243538 non-null  object
 6   from        243538 non-null  object
 7   time_taken  243538 non-null  object
 8   stop        243538 non-null  object
 9   arr_time    243538 non-null  object
 10  to          243538 non-null  object
 11  class       243538 non-null  object
 12  price       243538 non-null  int64 
dtypes: int64(2), object(11)
memory usage: 24.2+ MB
```

There were no null values in the train set. I also checked the test set. There were also no null values in the test set.

```
#이거 보면 1.01h m 이런거 있음
unique_times = pd.Series(train['time_taken'].unique())
print(unique_times.tail(10))
```
[54] ✓ 0.0s Python

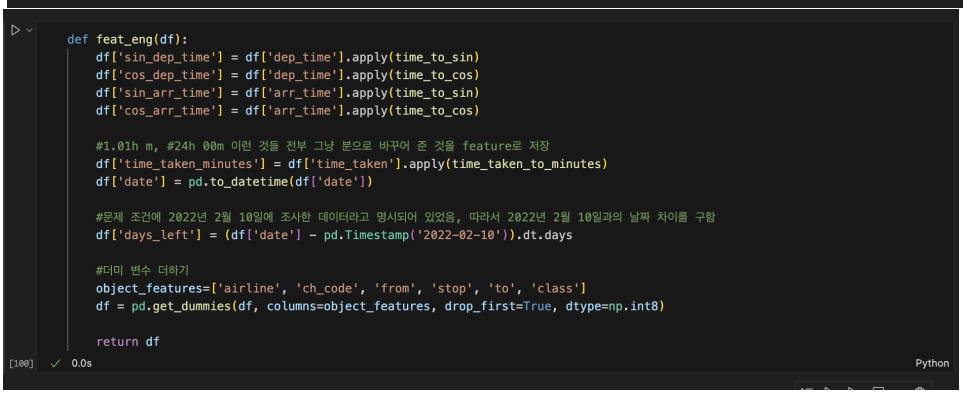
```
466    06h 59m
467     5h 30m
468    39h 50m
469    1.01h m
470    1.02h m
471    1.03h m
472    35h 40m
473    35h 20m
474    29h 55m
475    36h 35m
dtype: object
```

There were some strange form of timestamp form in train[‘time taken’], for example, 1.01h m. So I wanted to deal with this problem.



```python
def time_to_minutes(t):
    hours, minutes = map(int, t.split(":"))
    return hours * 60 + minutes


#대부분의 데이터는 24h 00m로 나오지만 1.01h m 과 같은 몇몇 결측치들을 봤음... 따라서 그거 처리하는 함수
def time_taken_to_minutes(t):
    #여분의 공백을 제거하고 소문자로 변환
    t = t.lower().strip()

    #"h"로 시간과 분을 분리
    parts = t.split('h')
    hours = 0
    minutes = 0

    #시간 부분 처리
    if parts[0]:
        hours = float(parts[0])

    #분 부분 처리
    if len(parts) > 1 and parts[1]:
        minutes_part = parts[1].strip().replace('m', '')  #'m' 문자를 제거
        if minutes_part:  #분 부분이 실제로 숫자를 포함하고 있는지 확인
            minutes = float(minutes_part)

    total_minutes = int(hours * 60 + minutes)

    return total_minutes

#교수님이 수업시간에 설명하셨듯이, 시간을 삼각함수에 매핑하면 모델 학습에 도움됨
def time_to_sin(t):
    return np.sin(2 * np.pi * time_to_minutes(t) / 1440)
def time_to_cos(t):
    return np.cos(2 * np.pi * time_to_minutes(t) / 1440)
```

[58] ✓ 0.0s Python



```python
def feat_eng(df):
    df['sin_dep_time'] = df['dep_time'].apply(time_to_sin)
    df['cos_dep_time'] = df['dep_time'].apply(time_to_cos)
    df['sin_arr_time'] = df['arr_time'].apply(time_to_sin)
    df['cos_arr_time'] = df['arr_time'].apply(time_to_cos)

    #1.01h m, #24h 00m 이런 것들 전부 그냥 분으로 바꾸어 준 것을 feature로 저장
    df['time_taken_minutes'] = df['time_taken'].apply(time_taken_to_minutes)
    df['date'] = pd.to_datetime(df['date'])

    #문제 조건에 2022년 2월 10일에 조사한 데이터라고 명시되어 있었음, 따라서 2022년 2월 10일과의 날짜 차이를 구함
    df['days_left'] = (df['date'] - pd.Timestamp('2022-02-10')).dt.days

    #더미 변수 더하기
    object_features=['airline', 'ch_code', 'from', 'stop', 'to', 'class']
    df = pd.get_dummies(df, columns=object_features, drop_first=True, dtype=np.int8)

    return df
```

[100] ✓ 0.0s Python

In my code, I first define a function ‘time to minutes’ to convert time into minutes for simplicity. To be more specific, ‘dep time’ and ‘arr time’ are object type, for example, 03:00 or 04:25. So this function converts them into just minutes.

I make a function called ’time taken to minutes’ to fix issues with how some ’time taken’ entries like ‘1.01h m’, as I mentioned before. I converted them into just minutes for easy and simple analysis.

Understanding the benefits of trigonometric time transformations as the professor Kim explained in class, I map departure and arrival times to sine and cosine functions, which can improve model training. I add these transformed times as new features.

I also used the date column to calculate the number of days after 2022-02-10.

Lastly, I added dummy variables for categorical features.

These are what I did for feature engineering in this mini kaggle.

```python
train = pd.read_csv('train.csv')
test=pd.read_csv('test.csv')
submission=pd.read_csv('submission.csv')

#train set feature engineering
processed_train = feat_eng(train)
X_tr = processed_train.drop(columns=['price', 'id', 'date', 'dep_time', 'arr_time','time_taken'])
y_tr = processed_train['price']

#test set feature engineering
processed_test = feat_eng(test)
X_test = processed_test.drop(columns=['id', 'date', 'dep_time', 'arr_time','time_taken'])
X_test = X_test.reindex(columns=X_tr.columns, fill_value=0)

#Data Scaling 이후 학습
dtr_pipeline = make_pipeline(StandardScaler(), DecisionTreeRegressor(random_state=0))
dtr_pipeline.fit(X_tr, y_tr)
y_pred = dtr_pipeline.predict(X_test)

#Dataframe 만든후, submission.csv 파일로 저장
submission = pd.DataFrame({'id': test['id'], 'price': y_pred})
submission.to_csv('submission.csv', index=False)
```

[98] ✓ 4.2s Python

This code performs feature engineering on both training and test datasets using ‘feat eng’ function which I made above.
I used StandardScalar() to standardize the data and trains a Decision Tree Regressor model. Finally, the model predicts prices for the test dataset and prepares a submission by saving the results to ‘submission.csv’.