ML_PS3

Clarice Tee

2025-02-20

```
import numpy as np
import pandas as pd
import os
import time
import statsmodels.api as sm
# Sci-kit learn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression,LinearRegression, Lasso
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import LeaveOneOut
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error, make_scorer
from mlxtend.feature_selection import ExhaustiveFeatureSelector as EFS
# Statsmodels
import statsmodels.api as sm

# Visualizations
import matplotlib.pyplot as plt
import seaborn as sns

# Iterator building blocks
# example: combinations('ABCD', 2) --> AB AC AD BC BD CD
from itertools import combinations

# Many were concerned with warnings. The short answer is that FutureWarning (most common)
# appears when a functionality is deprecated and will be replaced. Here's how to ignore t
# even if you should find a way to resolve them in a production environment.
import warnings
```

```python
warnings.filterwarnings("ignore", category=FutureWarning)
```

Part 1

1.a.

```python
# Load the dataset
directory = r"C:\Users\clari\OneDrive\Documents\Machine Learning\ps3"
default_path = os.path.join(directory, "Data-Default.csv")
default_df = pd.read_csv(default_path)
print(default_df.dtypes)
print(default_df.shape)
default_df.head()
```

```
default     object
student     object
balance    float64
income     float64
dtype: object
(10000, 4)
```

	default	student	balance	income
0	No	No	729.526495	44361.625074
1	No	Yes	817.180407	12106.134700
2	No	No	1073.549164	31767.138947
3	No	No	529.250605	35704.493935
4	No	No	785.655883	38463.495879

```python
na_count = pd.DataFrame(np.sum(default_df.isna(), axis = 0), columns = ["Count NAs"])
print(na_count)
```

```
         Count NAs
default          0
student          0
balance          0
income           0
```

```python
encoding_dict = {'Yes': 1,'No': 0}
for col in ['default', 'student']:
    default_df[col] = default_df[col].map(encoding_dict)

default_df.head()
```

	default	student	balance	income
0	0	0	729.526495	44361.625074
1	0	1	817.180407	12106.134700
2	0	0	1073.549164	31767.138947
3	0	0	529.250605	35704.493935
4	0	0	785.655883	38463.495879

getting a sense of the estimates of the coefficients as well as the standard errors associated with them.

```python
X = default_df[['balance', 'income']]
X = sm.add_constant(X)

y = default_df['default']

display(X.head(), y.head())
print(y.value_counts())
results = sm.Logit(y, X).fit()
print(results.summary())
```

	const	balance	income
0	1.0	729.526495	44361.625074
1	1.0	817.180407	12106.134700
2	1.0	1073.549164	31767.138947
3	1.0	529.250605	35704.493935
4	1.0	785.655883	38463.495879

```
0    0
1    0
2    0
3    0
4    0
Name: default, dtype: int64
```

```
default
0    9667
1     333
Name: count, dtype: int64
Optimization terminated successfully.
         Current function value: 0.078948
         Iterations 10
                           Logit Regression Results
==============================================================================
Dep. Variable:                default   No. Observations:                10000
Model:                          Logit   Df Residuals:                     9997
Method:                           MLE   Df Model:                            2
Date:                Thu, 20 Feb 2025   Pseudo R-squ.:                  0.4594
Time:                        22:47:52   Log-Likelihood:                -789.48
converged:                       True   LL-Null:                       -1460.3
Covariance Type:            nonrobust   LLR p-value:                4.541e-292
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        -11.5405      0.435    -26.544      0.000     -12.393     -10.688
balance        0.0056      0.000     24.835      0.000       0.005       0.006
income      2.081e-05   4.99e-06      4.174      0.000     1.1e-05    3.06e-05
==============================================================================

Possibly complete quasi-separation: A fraction 0.14 of observations can be
perfectly predicted. This might indicate that there is complete
quasi-separation. In this case some parameters will not be identified.
```

The SE for balance is 0 and 4.99e-06 for income

1.b.

```python
# Defining a function which performs the `logit` on each subsample
def boot_fn(data, index, constant=True, features=['balance','income'], target='default'):

    X = data[features].loc[index]
    if constant:
        X = sm.add_constant(X)
    y = data[target].loc[index]

    lr = sm.Logit(y,X).fit(disp=0)
    coefficients = [lr.params[0], lr.params[1], lr.params[2]]

    return coefficients
```

1.c.

```
# Define function to randomly select indices
def get_indices(data, n, seed):

    rng = np.random.default_rng(seed)    # allows you to set your seed
    indices = rng.choice(data.index,     # Use the dataset's indices as the input
                         int(n),         # Number of indices per sample
                         replace=True    # Draw samples with replacement
                        )
    return indices
```

```
# Fundtion to execute it
def boot(data, func, B):
    '''
    Executing a bootstrap over B subsamples
    to estimate the (mean of) coefficients
    and their associated standard errors.

    Inputs:
        - data (pd.Dataframe): data to sample from
        - func (fn): function executing the regression
        - B (int): number of subsamples

    Ouput:
        - restults (dict of dicts): bootstrapped coefficients
            and the associated standard errors
    '''
    # Step 4
    coef_intercept = []
    coef_balance = []
    coef_income = []

    coefs = ['intercept', 'balance', 'income']
    output = {coef: [] for coef in coefs}
    for i in range(B):
        # set new seed (=i) every time you get a new subsample
        reg_out = func(data, get_indices(data, len(data), i))
        for i, coef in enumerate(coefs):
            output[coef].append(reg_out[i])

    # Step 5
    results = {}
```

```
    for coef in coefs:
        results[coef] = {
            'estimate': np.mean(output[coef]),
            'std_err': np.std(output[coef])
        }

    return results
```

```
# Run the bootstrap
results = boot(default_df, boot_fn, 1000)
for i, x in results.items():
    print(
        f"{i.capitalize()}: \n\tEstimate: {x['estimate']}\n\tStandard Error: {x['std_err
```

```
Intercept:
    Estimate: -11.599221788148167
    Standard Error: 0.4407660085830434
Balance:
    Estimate: 0.005675350638457242
    Standard Error: 0.0002310150611592383
Income:
    Estimate: 2.1092285173232854e-05
    Standard Error: 4.836558867887688e-06
```

1.d. the estimated SE's using the bootstrap method and the logit one in a are quite close.

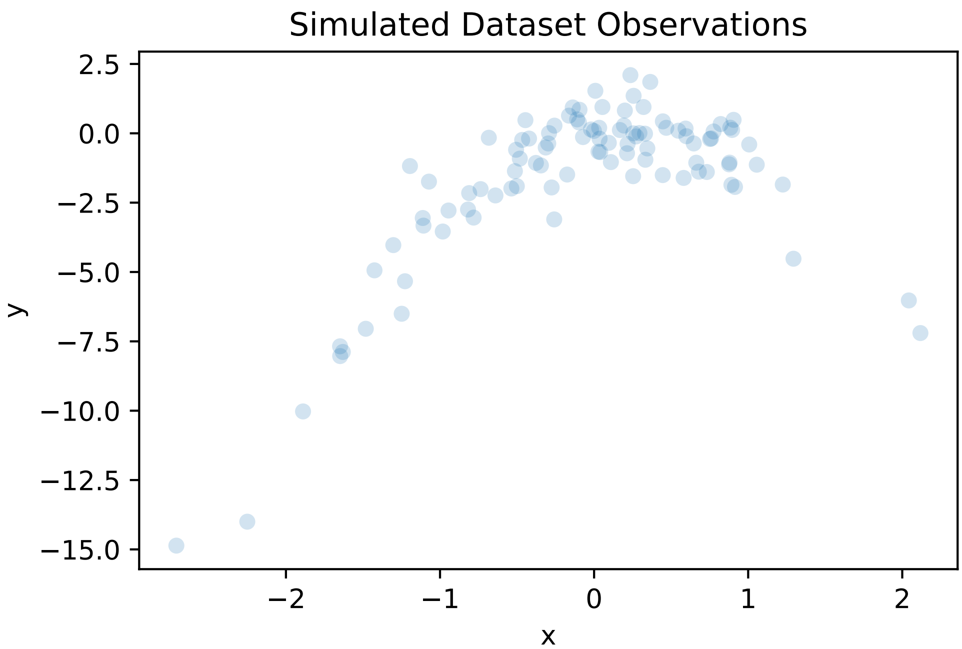
Part 2

2.a.

```
# Generate a simulated dataset
rng = np.random.default_rng(1)
x = rng.normal(size=100)
y = x - 2 * x**2 + rng.normal(size=100)
```

2.b.

```
# Scatter plot X and Y with fitted line
fig, ax = plt.subplots()
ax.scatter(x, y,
alpha=0.2, s=25)
ax.set_title("Simulated Dataset Observations")
ax.set_xlabel("x")
ax.set_ylabel("y")
plt.show()
```



INterpretation

2.c.

```
# Set random seed
np.random.seed(10)

simulated_df = pd.DataFrame({"x": x, "y": y})
for i in range(2, 5):
    simulated_df[f'x{i}'] = simulated_df['x']**i

models = [['x'], ['x', 'x2'], ['x', 'x2', 'x3'], ['x', 'x2', 'x3', 'x4']]

for i, model in enumerate(models, 1):
    X = simulated_df[model]
```

```
    y = simulated_df['y']

    lr = LinearRegression()
    scores = cross_val_score(lr, X, y, cv=LeaveOneOut(), scoring='neg_mean_squared_error
    loocv_error = -scores.mean()

    print(f"Model {i} LOOCV Error: {loocv_error:.6f}")
```

```
Model 1 LOOCV Error: 6.633030
Model 2 LOOCV Error: 1.122937
Model 3 LOOCV Error: 1.301797
Model 4 LOOCV Error: 1.332394
```

2.d. There was no change in the results versus those in part c when using a different random seed. This is because inLOOCV, the validation set consists of only one observation, and the process is repeated for all n data points. Since the splitting of data into training and validation sets are without replacement (each observation is used exactly once as the validation set), randomization isn't needed, so changing the random seed din't change the estimates.

```
# Set new random seed
np.random.seed(20)

# Create df of simulated data set
simulated_df = pd.DataFrame({"y": y,
    "x": x})

# Add features
simulated_df["x2"] = simulated_df["x"] ** 2
simulated_df["x3"] = simulated_df["x"] ** 3
simulated_df["x4"] = simulated_df["x"] ** 4

# Prepare list to hold squared errors
squared_errors = {i: [] for i in range(1, 5)}

# Iterate over models
for model_num in range(1, 5):
    for index in simulated_df.index:
        # Split training and validation set
        train = simulated_df.iloc[simulated_df.index != index, :]
        valid = simulated_df.iloc[simulated_df.index == index, :]
```

```python
        # Select features
        if model_num == 1:
            features = ["x"]
        elif model_num == 2:
            features = ["x", "x2"]
        elif model_num == 3:
            features = ["x", "x2", "x3"]
        else:
            features = ["x", "x2", "x3", "x4"]

        # Fit model and calculate mse
        ols = LinearRegression().fit(train[features], train[["y"]])
        valid_pred = ols.predict(valid[features])
        valid_actual = valid[["y"]]
        squared_error = np.power(valid_pred - valid_actual, 2)

        # Store squared error
        squared_errors[model_num].append(squared_error.values[0][0])

# Calculate mean squared errors (MSE)
for model_num in range(1, 5):
    print(f'MSE for model {model_num} using LOOCV: {round(np.mean(squared_errors[model_nu
```

```
MSE for model 1 using LOOCV: 6.63303
MSE for model 2 using LOOCV: 1.12294
MSE for model 3 using LOOCV: 1.3018
MSE for model 4 using LOOCV: 1.33239
```

2.e. Model 2 had the smallest LOOCV error of 1.12294, confirming that it most closely approximates the true underlying relationship in the data. This result aligns with our expectations because the data was generated using a quadratic relationship between x and yfrom n 2a. Since the true data-generating process is quadratic, a quadratic model i can better capture this relationship compared to the other models.

2.f.

```python
# Define the function to fit models and display summary
def model_fit(predictors):
    """
    Fitting each model on simulated_df and display the summary
    Inputs:
        - predictors (list): predictors to include in the linear regression model.
```

```python
    Output:
        - Summary of the model
    """
    # Split the data into target and predictors
    X = simulated_df[predictors]
    X = sm.add_constant(X)
    y = simulated_df["y"]

    # Fit the model to the data
    ols = sm.OLS(y, X).fit()

    # Display summary
    print(ols.summary())

# Define models as a dictionary
models = {
    1: ["x"],
    2: ["x", "x2"],
    3: ["x", "x2", "x3"],
    4: ["x", "x2", "x3", "x4"]
}

# Fit models and display summaries
print("\nModel Summaries:")
for model_num, predictors in models.items():
    print(f"\nModel {model_num}:")
    model_fit(predictors)
```

```
Model Summaries:

Model 1:
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.318
Model:                            OLS   Adj. R-squared:                  0.311
Method:                 Least Squares   F-statistic:                     45.60
Date:                Thu, 20 Feb 2025   Prob (F-statistic):           1.04e-09
Time:                        22:48:18   Log-Likelihood:                -230.83
No. Observations:                 100   AIC:                             465.7
Df Residuals:                      98   BIC:                             470.9
Df Model:                           1
```

```
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -1.4650      0.247     -5.937      0.000      -1.955      -0.975
x              1.9494      0.289      6.752      0.000       1.376       2.522
==============================================================================
Omnibus:                       52.788   Durbin-Watson:                   1.972
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              149.089
Skew:                          -1.953   Prob(JB):                     4.22e-33
Kurtosis:                       7.530   Cond. No.                         1.20
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi

Model 2:
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.887
Model:                            OLS   Adj. R-squared:                  0.884
Method:                 Least Squares   F-statistic:                     379.5
Date:                Thu, 20 Feb 2025   Prob (F-statistic):           1.36e-46
Time:                        22:48:18   Log-Likelihood:                -141.06
No. Observations:                 100   AIC:                             288.1
Df Residuals:                      97   BIC:                             295.9
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0728      0.119     -0.611      0.543      -0.309       0.164
x              0.9663      0.126      7.647      0.000       0.715       1.217
x2            -2.0047      0.091    -22.072      0.000      -2.185      -1.824
==============================================================================
Omnibus:                        1.338   Durbin-Watson:                   2.197
Prob(Omnibus):                  0.512   Jarque-Bera (JB):                0.814
Skew:                           0.119   Prob(JB):                        0.666
Kurtosis:                       3.372   Cond. No.                         2.23
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
```

```
Model 3:
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.888
Model:                            OLS   Adj. R-squared:                  0.885
Method:                 Least Squares   F-statistic:                     253.8
Date:                Thu, 20 Feb 2025   Prob (F-statistic):           1.70e-45
Time:                        22:48:18   Log-Likelihood:                -140.47
No. Observations:                 100   AIC:                             288.9
Df Residuals:                      96   BIC:                             299.4
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0572      0.120     -0.477      0.635      -0.295       0.181
x              1.1146      0.187      5.945      0.000       0.742       1.487
x2            -2.0471      0.099    -20.673      0.000      -2.244      -1.851
x3            -0.0643      0.060     -1.070      0.287      -0.184       0.055
==============================================================================
Omnibus:                        0.845   Durbin-Watson:                   2.199
Prob(Omnibus):                  0.655   Jarque-Bera (JB):                0.392
Skew:                           0.052   Prob(JB):                        0.822
Kurtosis:                       3.289   Cond. No.                         5.95
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifie

Model 4:
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.894
Model:                            OLS   Adj. R-squared:                  0.890
Method:                 Least Squares   F-statistic:                     200.2
Date:                Thu, 20 Feb 2025   Prob (F-statistic):           2.22e-45
Time:                        22:48:18   Log-Likelihood:                -137.74
No. Observations:                 100   AIC:                             285.5
Df Residuals:                      95   BIC:                             298.5
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
```

```
                  coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const           0.1008      0.136      0.743      0.460      -0.169       0.370
x               0.9050      0.205      4.423      0.000       0.499       1.311
x2             -2.5059      0.221    -11.336      0.000      -2.945      -2.067
x3              0.0338      0.073      0.466      0.642      -0.110       0.178
x4              0.1042      0.045      2.309      0.023       0.015       0.194
==============================================================================
Omnibus:                        2.476   Durbin-Watson:                   2.163
Prob(Omnibus):                  0.290   Jarque-Bera (JB):                2.097
Skew:                           0.118   Prob(JB):                        0.351
Kurtosis:                       3.669   Cond. No.                         19.9
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifie
```

```
# Display the results
print("model i:\n")
model_fit(["x"])
```

```
model i:

                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.318
Model:                            OLS   Adj. R-squared:                  0.311
Method:                 Least Squares   F-statistic:                     45.60
Date:                Thu, 20 Feb 2025   Prob (F-statistic):           1.04e-09
Time:                        22:48:18   Log-Likelihood:                -230.83
No. Observations:                 100   AIC:                             465.7
Df Residuals:                      98   BIC:                             470.9
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          -1.4650      0.247     -5.937      0.000      -1.955      -0.975
x               1.9494      0.289      6.752      0.000       1.376       2.522
==============================================================================
Omnibus:                       52.788   Durbin-Watson:                   1.972
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              149.089
Skew:                          -1.953   Prob(JB):                     4.22e-33
```

```
Kurtosis:                       7.530   Cond. No.                         1.20
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifie
```

```
print("\nmodel ii:\n")
model_fit(["x", "x2"])
```

```

model ii:

                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.887
Model:                            OLS   Adj. R-squared:                  0.884
Method:                 Least Squares   F-statistic:                     379.5
Date:                Thu, 20 Feb 2025   Prob (F-statistic):           1.36e-46
Time:                        22:48:18   Log-Likelihood:                -141.06
No. Observations:                 100   AIC:                             288.1
Df Residuals:                      97   BIC:                             295.9
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0728      0.119     -0.611      0.543      -0.309       0.164
x              0.9663      0.126      7.647      0.000       0.715       1.217
x2            -2.0047      0.091    -22.072      0.000      -2.185      -1.824
==============================================================================
Omnibus:                        1.338   Durbin-Watson:                   2.197
Prob(Omnibus):                  0.512   Jarque-Bera (JB):                0.814
Skew:                           0.119   Prob(JB):                        0.666
Kurtosis:                       3.372   Cond. No.                         2.23
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifie
```

```
print("\nmodel iii:\n")
model_fit(["x", "x2", "x3"])
```

```
model iii:

                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.888
Model:                            OLS   Adj. R-squared:                  0.885
Method:                 Least Squares   F-statistic:                     253.8
Date:                Thu, 20 Feb 2025   Prob (F-statistic):           1.70e-45
Time:                        22:48:18   Log-Likelihood:                -140.47
No. Observations:                 100   AIC:                             288.9
Df Residuals:                      96   BIC:                             299.4
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0572      0.120     -0.477      0.635      -0.295       0.181
x              1.1146      0.187      5.945      0.000       0.742       1.487
x2            -2.0471      0.099    -20.673      0.000      -2.244      -1.851
x3            -0.0643      0.060     -1.070      0.287      -0.184       0.055
==============================================================================
Omnibus:                        0.845   Durbin-Watson:                   2.199
Prob(Omnibus):                  0.655   Jarque-Bera (JB):                0.392
Skew:                           0.052   Prob(JB):                        0.822
Kurtosis:                       3.289   Cond. No.                         5.95
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifie
```

```
print("\nmodel iv:\n")
model_fit(["x", "x2", "x3", "x4"])
```

```
model iv:

                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.894
Model:                            OLS   Adj. R-squared:                  0.890
Method:                 Least Squares   F-statistic:                     200.2
Date:                Thu, 20 Feb 2025   Prob (F-statistic):           2.22e-45
```

```
Time:                        22:48:18   Log-Likelihood:                -137.74
No. Observations:                 100   AIC:                             285.5
Df Residuals:                      95   BIC:                             298.5
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1008      0.136      0.743      0.460      -0.169       0.370
x              0.9050      0.205      4.423      0.000       0.499       1.311
x2            -2.5059      0.221    -11.336      0.000      -2.945      -2.067
x3             0.0338      0.073      0.466      0.642      -0.110       0.178
x4             0.1042      0.045      2.309      0.023       0.015       0.194
==============================================================================
Omnibus:                        2.476   Durbin-Watson:                   2.163
Prob(Omnibus):                  0.290   Jarque-Bera (JB):                2.097
Skew:                           0.118   Prob(JB):                        0.351
Kurtosis:                       3.669   Cond. No.                         19.9
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifie
```

The p-values in each models show that, while linear ( ) and quadratic (2) associations are statistically significant at 1% significance level, predictors of higher degree (3, 4) are not, which agree with the conclusions drawn based on the CV results.

Part 3

3.a.

```python
directory = r"C:\Users\clari\OneDrive\Documents\Machine Learning\ps3"
boston_path = os.path.join(directory, "Boston.csv")
boston_df = pd.read_csv(boston_path)
print(boston_df.dtypes)
print(boston_df.shape)
boston_df.head()
X = boston_df.drop("CRIM", axis=1)
y = boston_df["CRIM"]
```

```
CRIM       float64
```

```
ZN          float64
INDUS       float64
CHAS        float64
NOX         float64
RM          float64
AGE         float64
DIS         float64
RAD         float64
TAX         float64
PTRATIO     float64
B           float64
LSTAT       float64
MDEV        float64
dtype: object
(506, 14)
```

```
# Set up cross-validation
kfold = KFold(n_splits=5, shuffle=True, random_state=24)


# Make RMSE scorer
def rmse(y_true, y_pred):
    return sqrt(mean_squared_error(y_true, y_pred))
rmse_scorer = make_scorer(
    rmse,
    greater_is_better=False
)
```

BEST SUBSET

```
# Create a custom RMSE scorer
def rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))

rmse_scorer = make_scorer(rmse, greater_is_better=False)

# Set up cross-validation
kfcv = KFold(n_splits=5, shuffle=True, random_state=24)

# Set up the model
model = LinearRegression()

# Start timing
```

```python
start_time = time.time()

# Perform best subset selection
efs = EFS(
    model,
    min_features=1,
    max_features=13,  # Boston dataset has 13 features excluding CRIM
    scoring=rmse_scorer,
    cv=kfcv,
    n_jobs=-1
)

efs.fit(X, y)

# Find the best subset
best_subset = max(efs.subsets_.values(), key=lambda x: x["avg_score"])
best_features = best_subset["feature_names"]
best_rmse = -best_subset["avg_score"]

# End timing
end_time = time.time()

print(f"Best Model Features: {best_features}")
print(f"Best Model RMSE: {round(best_rmse, 5)}")
print(f"Execution Time: {round(end_time - start_time, 2)} seconds")
```

```
Features: 1/8191Features: 2/8191Features: 3/8191Features: 4/8191Features: 5/8191Features

Best Model Features: ('ZN', 'INDUS', 'CHAS', 'NOX', 'DIS', 'RAD', 'B', 'LSTAT')
Best Model RMSE: 6.04235
Execution Time: 69.92 seconds
```

```python
print("\nBest Model from Best Subset Approach:")
print(best_features)
print(f"\nBest Model RMSE: {round(best_rmse, 5)}")
end_time = time.time()
print(f"Running Time: {end_time - start_time}")
```

```
Best Model from Best Subset Approach:
('ZN', 'INDUS', 'CHAS', 'NOX', 'DIS', 'RAD', 'B', 'LSTAT')
```

```
Best Model RMSE: 6.04235
Running Time: 71.97150897979736
```

FORWARD STEPWISE

```
from sklearn.feature_selection import SequentialFeatureSelector
start_time = time.time()
sfs_forward = SequentialFeatureSelector(
    ols,
    n_features_to_select="auto",
    direction="forward",
    scoring=rmse_scorer,  # Use RMSE instead of R²
    cv=kfcv
)

# Fit the feature selector
sfs_forward.fit(X, y)
# Extract selected features
forward_selected_features = X.columns[sfs_forward.get_support()].tolist()
```

```
print("\nThe best model from the Forward Stepwise Approach is:")
print(forward_selected_features)
best_model_rmse = -cross_val_score(
    ols,
    X[forward_selected_features],
    y,
    cv=kfcv,
    scoring=rmse_scorer
).mean()
print(f"\nThe best model RMSE: {round(best_model_rmse, 5)}")
end_time = time.time()
print(f"Running Time: {end_time - start_time}")
```

```
The best model from the Forward Stepwise Approach is:
['ZN', 'NOX', 'DIS', 'RAD', 'B', 'LSTAT']

The best model RMSE: 6.04652
Running Time: 0.9751005172729492
```

BACKWARD STEPWISE

```python
# Backward Stepwise
start_time = time.time()
sfs_backward = SequentialFeatureSelector(
ols,
n_features_to_select="auto",
direction="backward",
scoring=rmse_scorer,
cv=kfcv
)
```

```python
# Fit the feature selector
sfs_backward.fit(X, y)
# Extract selected features
backward_selected_features = X.columns[sfs_backward.get_support()].tolist()
```

```python
print("\nBest Model from Backward Stepwise Approach:")
print(backward_selected_features)
# Evaluate RMSE of the best model
best_model_rmse = -cross_val_score(
ols,
X[backward_selected_features],
y,
cv=kfcv,
scoring=rmse_scorer
).mean()
print(f"\nBest Model RMSE: {round(best_model_rmse, 5)}")
end_time = time.time()
print(f"Running Time: {end_time - start_time}")
```

```
Best Model from Backward Stepwise Approach:
['ZN', 'INDUS', 'NOX', 'DIS', 'RAD', 'B', 'LSTAT']

Best Model RMSE: 6.04247
Running Time: 1.0645697116851807
```

RMSE: Based on the RMSE, the best predictive power to wrost is: Best Subset > Backward Stepwise > Forward Stepwise. The variation in predictive power is liekly influenced by which predictors are included or excluded. Forward Stepwise model excludes predictors like "INDUS," while both stepwise approaches exclude "CHAS."

Running time: Because of how thororugh the Best Subset approach is, it takes much longer to run–a whole minute more than the stepwise methods, even with only 13 predictors.The Stepwise models are more efficient, making them preferable when resources or time are limited.

3.b.

```
boston_df = boston_df.reset_index(drop=True)
X = boston_df.drop("CRIM", axis=1)
y = boston_df["CRIM"].copy()
```

Best Subset Model

```
best_subset_features = ['ZN', 'INDUS', 'CHAS', 'NOX', 'DIS', 'RAD', 'B', 'LSTAT']
X_bs = sm.add_constant(boston_df[best_subset_features])
model_bs = sm.OLS(y, X_bs).fit()

# Forward Stepwise Model
forward_features = ['ZN', 'NOX', 'DIS', 'RAD', 'B', 'LSTAT']
X_fs = sm.add_constant(boston_df[forward_features])
model_fs = sm.OLS(y, X_fs).fit()

# Backward Stepwise Model
backward_features = ['ZN', 'INDUS', 'NOX', 'DIS', 'RAD', 'B', 'LSTAT']
X_bw = sm.add_constant(boston_df[backward_features])
model_bw = sm.OLS(y, X_bw).fit()
```

5fCV

```
# five_fcv function
def five_fcv(name, predictors):
    """Calculate 5-Fold CV RMSE for specified predictors"""
    X = boston_df[predictors]
    y = boston_df["CRIM"]

    lr = LinearRegression()
    cv = KFold(n_splits=5, shuffle=True, random_state=24)

    # Calculate  MSE and RMSE
    mse_scores = -cross_val_score(lr, X, y,
                                cv=cv,
                                scoring='neg_mean_squared_error')
    rmse_scores = np.sqrt(-cross_val_score(lr, X, y,
```

```
                                             cv=cv,
                                             scoring='neg_mean_squared_error'))

    print(f"{name} Approach:")
    print(f"- 5FCV MSE: {np.mean(mse_scores):.2f}")
    print(f"- 5FCV RMSE: {np.mean(rmse_scores):.2f}\n")
```

Evaluating

```
five_fcv("Best Subset", best_subset_features)
five_fcv("Forward Stepwise", forward_features)
five_fcv("Backward Stepwise", backward_features)
```

```
Best Subset Approach:
- 5FCV MSE: 42.92
- 5FCV RMSE: 6.04

Forward Stepwise Approach:
- 5FCV MSE: 43.04
- 5FCV RMSE: 6.05

Backward Stepwise Approach:
- 5FCV MSE: 42.97
- 5FCV RMSE: 6.04
```

AIC- asked phoenix how to do AIC calculation in python

```
print("AIC Values")
print(f"Best Subset: {model_bs.aic:.1f}")
print(f"Forward Stepwise: {model_fs.aic:.1f}")
print(f"Backward Stepwise: {model_bw.aic:.1f}")


print(" Key Findings")
print("Best model by 5FCV: Best Subset (lowest MSE)")
print("Best model by AIC: Forward Stepwise (lowest AIC)")
```

```
AIC Values
Best Subset: 3341.0
Forward Stepwise: 3339.3
Backward Stepwise: 3340.2
```

```
 Key Findings
Best model by 5FCV: Best Subset (lowest MSE)
Best model by AIC: Forward Stepwise (lowest AIC)
```

The Best Subset approach had the lowest 5fCV MSE (42.92), demonstrating superior predictive performance compared to Forward Stepwise (43.04) and Backward Stepwise (42.97) methods. But the Forward Stepwise model showed the best AIC value (3339.3), indicating better balance between model fit and complexity.

According to the book (after asking Phoenix: why would one model be better in terms of MSE but another at AIC? what is the reasoning behind it?), a model may have a lower MSE due to its ability to fit the training data very well, but it may not necessarily perform better on unseen data. Meanwhile, the AIC adjusts for this by incorporating a penalty for the number of parameters to avoid overfitting, thus sometimes selecting simpler models that generalize better despite potentially higher MSE on training data.

Why they select different models: Because the Best Subset method searchrd through all predictor combinations (8 features), it is liekly that it has a stronger predictive performance, though at greater computational cost. Forward Stepwise builds models sequentially by adding features, potentially missing important combos, while Backward Stepwise removes features from a full model, preserving more predictors.

5FCV is preferred over training error since it prevents overfitting through repeated data splitting, providing more reliable estimates of real-world performance. AIC remains valuable for model comparison with limited data as it penalizes complexity (2k/n penalty term). The metrics may not align because 5FCV focuses on predictive accuracy while AIC estimates information loss relative to the true data-generating process.

3.c. NO, the selected models don't use all 13 features. Excluded predictors were AGE, TAX, and PTRATIO.

This is bececause the model tries to reduce overvitting by removing variables that don't add much to the model in terms of predictive power, which in itself helps us avoid the problem of multicollinearity. This thus makes the model outcome easier to interpret.