Instructions

To do this assignment, save a copy to your own Google drive by going to:

> File --> Save a copy in Drive

Before working on this notebook, please disable inline AI suggestions by going to:

> Settings (top right gear icon) --> AI assistance --> Show AI-powered inline completions (uncheck)

As stated in the syllabus, you're welcome to use AI as an aid, but your answers should be your own.

Submission instructions:

- **Due**: Wednesday 11/12 at 11:59PM AOE
- **Submission**: Turn in both a printed **PDF** and the **editable share link** on MyCourses.

Remember that to make the share link editable, you must **convert the notebook to "anyone with a link can edit"** before copying and pasting the link.

Problem 1

In this problem, we will investigate the effect of a linear transformation of features on basic regression models.

Suppose that we have a linear regression problem with n observations and m linearly-independent features (including the intercept). The design matrix in this linear regression is $X \in \mathbb{R}^{n \times m}$ and the target vector is $y \in \mathbb{R}^n$. A linear transformation of features transforms the design matrix X to a new design matrix XB for some matrix $B \in \mathbb{R}^{m \times m}$. With this setup, please complete the following steps.

Problem 1.1

Create or reuse a *simple* dataset. Six observations and three features are sufficient. Construct the design matrix X and the target vector y for this dataset. Also construct a linear transformation matrix B such that B is **not** diagonal but is invertible (B^{-1} exists). Compute the best linear regression fits for the original features and transformed features in this simple example and discuss whether they are the same or not. The solution to this question should be R or Python code.

```
import numpy as np
from sklearn.linear_model import LinearRegression

# --- Step 1: Create a simple dataset ---
# 6 observations, 3 features (including intercept column)
X = np.array([
    [1, 0.5, 1.2],
    [1, 1.0, 0.8],
    [1, 1.5, 1.5],
    [1, 2.0, 1.0],
    [1, 2.5, 2.2],
    [1, 3.0, 2.8]
```

```
])
y = np.array([2.0, 2.5, 3.0, 3.2, 4.1, 4.8])

# --- Step 2: Construct a linear transformation matrix B ---
# B is 3x3, invertible, and not diagonal
B = np.array([
    [1, 1, 0],
    [0, 1, 1],
    [0, 0, 1]
])
assert np.linalg.det(B) != 0, "B must be invertible"

# --- Step 3: Apply the feature transformation ---
X_transformed = X @ B

# --- Step 4: Fit linear regression models ---
model_original = LinearRegression(fit_intercept=False)  # intercept already in X
model_transformed = LinearRegression(fit_intercept=False)

model_original.fit(X, y)
model_transformed.fit(X_transformed, y)

# --- Step 5: Compare coefficients ---
print("Original coefficients (β):")
print(model_original.coef_)

print("\nTransformed coefficients (γ):")
print(model_transformed.coef_)

# --- Step 6: Relate the coefficients ---
# Since X' = X B, we have: y ≈ X B γ = X β  ⇒  β = B γ
beta_from_gamma = B @ model_transformed.coef_
print("\nB * γ (should match β):")
print(beta_from_gamma)

# --- Step 7: Verify predictions are the same ---
y_pred_orig = model_original.predict(X)
y_pred_trans = model_transformed.predict(X_transformed)

print("\nAre predictions identical?", np.allclose(y_pred_orig, y_pred_trans))
```

```
Original coefficients (β):
[1.22343427 0.85403423 0.34653   ]

Transformed coefficients (γ):
[0.71593004 0.50750423 0.34653   ]
```

Problem 1.2

Prove that whenever $(X^T X)^{-1}$ and B^{-1} exist, then the linear transformation of features does not affect the *prediction* of the best linear regresion fit. *Hint*: The following identity may be useful: $(B^T A^T AB)(B^{-1}(A^T A)^{-1}(B^T)^{-1}) = I$ where I is the identity matrix.

For the original features X:

$\hat{\beta} = (X^\top X)^{-1} X^\top y$

And for the transformed features, since $X_B := XB$:

$\hat{\gamma} = ((XB)^\top (XB))^{-1}(XB)^\top y = (B^\top X^\top XB)^{-1} B^\top X^\top y$

Since

$(B^\top X^\top XB)(B^{-1}(X^\top X)^{-1}(B^\top)^{-1} = I$

And

$(B^\top X^\top XB)^{-1}((B^\top X^\top XB)(B^{-1}(X^\top X)^{-1}(B^\top)^{-1}) = (B^\top X^\top XB)^{-1} I$

Then

$(B^{-1}(X^\top X)^{-1}(B^\top)^{-1}) = (B^\top X^\top XB)^{-1}$

Multiplying by $B^\top$:

$(B^\top X^\top XB)^{-1} B^\top = (B^\top)^{-1}(X^\top X)^{-1} B^{-1} B^\top = (X^\top X)^{-1} B^{-1}$

so $\hat{\gamma}$ becomes

$\hat{\gamma} = (X^\top X)^{-1} B^{-1} X^\top y$

substituting in $\hat{\beta}$

$\hat{\gamma} = \hat{\beta} B^{-1}$

and it follows

$XB\hat{\gamma} = XB(B^{-1}\hat{\beta}) = X\hat{\beta}$

Problem 1.3

Give an example that proves that when B^{-1} does not exist, the predictions of the linear regression can be affected by the linear transformation. The example should be *very* simple.

```
# --- Step 1: Create a simple dataset ---
# 6 observations, 3 features (including intercept column)
X = np.array([
    [1, 0],
    [1, 1],
    [1, 2]
])
y = np.array([1, 2, 4])

# --- Step 2: Construct a linear transformation matrix B ---
# B is 3x3, invertible, and not diagonal
B = np.array([
    [1, 1],
    [1, 1],
])

# --- Step 3: Apply the feature transformation ---
X_transformed = X @ B

# --- Step 4: Fit linear regression models ---
```

```
model_original = LinearRegression(fit_intercept=False)  # intercept already in X
model_transformed = LinearRegression(fit_intercept=False)

model_original.fit(X, y)
model_transformed.fit(X_transformed, y)

# --- Step 5: Compare coefficients ---
print("Original coefficients (β):")
print(model_original.coef_)

print("\nTransformed coefficients (γ):")
print(model_transformed.coef_)

y_pred_orig = model_original.predict(X)
y_pred_trans = model_transformed.predict(X_transformed)

print("Prediction from original features")
print(y_pred_orig)
print("Prediction from Transposed features")
print(y_pred_trans)
```

```
Original coefficients (β):
[0.83333333 1.5       ]

Transformed coefficients (γ):
[0.60714286 0.60714286]
Prediction from original features
[0.83333333 2.33333333 3.83333333]
Prediction from Transposed features
[1.21428571 2.42857143 3.64285714]
```

Problem 1.4

Does the linear transformation described above affects the predictions of ridge regression? Prove your claim.

Yes, the proof from problem 1.2 still follows if we consider the ridge prediction $\hat{\beta} = (X^\top X + \lambda I)^{-1} X^\top y$ and coresponding $\hat{\gamma} = (B^\top X^\top X B + \lambda I)^{-1} X^\top y$

Problem 2

Problem 2.1

Implement your own cubic spline with at least 3 knots (not using a library) and use it with linear regression on a simple problem of your choice with a single feature. Plot the fitted function.

Problem 2.2

Compare your predictions of your splines with the built-in cubic spline in R or Python using the same knots. Do you get the same results? Discuss why or why not.

Problem 3

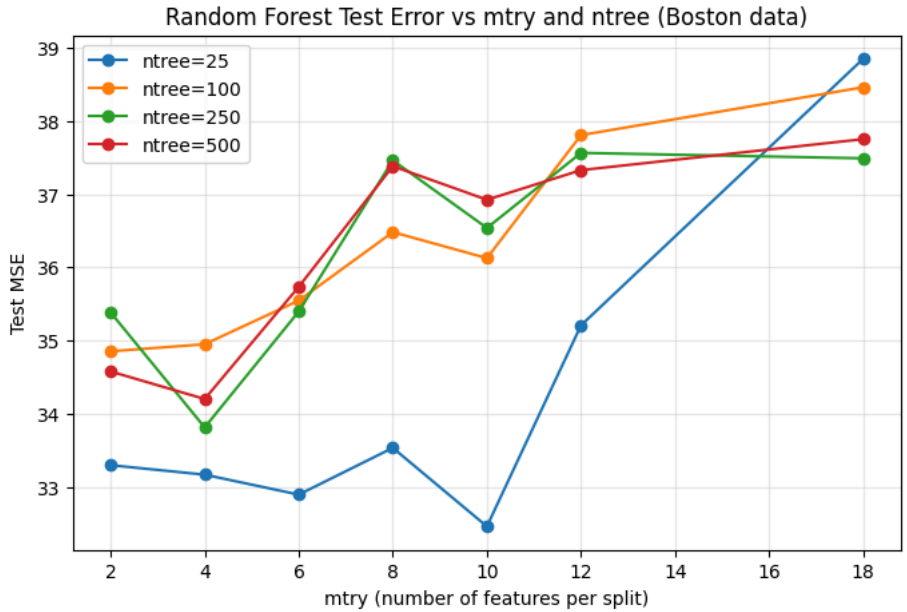
The lab in Chapter 8 applies random forests to the `Boston` data using `mtry = 6` and using `ntree = 25` and `ntree = 500`. Create a plot displaying the test error resulting from random forests on this data set for numerous values of `mtry` and `ntree`. You can model your plot after Figure 8.10 in Chapter 8 in ISRL. Briefly describe the results obtained.

Below is some starter code to load the Boston dataset in Pandas. See here for a discussion of some ethical issues with this dataset.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

data_url = "https://lib.stat.cmu.edu/datasets/boston_corrected.txt"
boston_df = pd.read_csv(data_url, sep="\t", skiprows=8, header=0, encoding='latin1')
display(boston_df)


X = boston_df.drop(["OBS.", "TOWN", "CRIM"], axis=1)
y = boston_df.CRIM
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

mtry_values = [2, 4, 6, 8, 10, 12, X.shape[1]]  # number of features per split
ntree_values = [25, 100, 250, 500]

test_errors = np.zeros((len(ntree_values), len(mtry_values)))

for i, ntree in enumerate(ntree_values):
    for j, mtry in enumerate(mtry_values):
        rf = RandomForestRegressor(
            n_estimators=ntree,
            max_features=mtry,
            random_state=42,
            n_jobs=-1
        )
        rf.fit(X_train, y_train)
        y_pred = rf.predict(X_test)
        test_errors[i, j] = mean_squared_error(y_test, y_pred)

plt.figure(figsize=(8,5))
for i, ntree in enumerate(ntree_values):
    plt.plot(mtry_values, test_errors[i, :], marker='o', label=f'ntree={ntree}')
plt.xlabel('mtry (number of features per split)')
plt.ylabel('Test MSE')
plt.title('Random Forest Test Error vs mtry and ntree (Boston data)')
plt.legend()
plt.grid(alpha=0.3)
plt.show()
df = pd.DataFrame(test_errors, index=[f'ntree={n}' for n in ntree_values], columns=mtry_values)
print("Test MSE (rows=ntree, columns=mtry):")
print(df.round(2))
```

	OBS.	TOWN	TOWN#	TRACT	LON	LAT	MEDV	CMEDV	CRIM	ZN	...	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	1	Nahant	0	2011	-70.9550	42.2550	24.0	24.0	0.00632	18.0	...	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	2	Swampscott	1	2021	-70.9500	42.2875	21.6	21.6	0.02731	0.0	...	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	3	Swampscott	1	2022	-70.9360	42.2830	34.7	34.7	0.02729	0.0	...	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03
3	4	Marblehead	2	2031	-70.9280	42.2930	33.4	33.4	0.03237	0.0	...	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
4	5	Marblehead	2	2032	-70.9220	42.2980	36.2	36.2	0.06905	0.0	...	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	502	Winthrop	91	1801	-70.9860	42.2312	22.4	22.4	0.06263	0.0	...	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.67
502	503	Winthrop	91	1802	-70.9910	42.2275	20.6	20.6	0.04527	0.0	...	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08
503	504	Winthrop	91	1803	-70.9948	42.2260	23.9	23.9	0.06076	0.0	...	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64
504	505	Winthrop	91	1804	-70.9875	42.2240	22.0	22.0	0.10959	0.0	...	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48
505	506	Winthrop	91	1805	-70.9825	42.2210	11.9	19.0	0.04741	0.0	...	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.88

506 rows × 21 columns




```
Test MSE (rows=ntree, columns=mtry):
              2      4      6      8      10     12     18
ntree=25   33.30  33.17  32.89  33.53  32.46  35.21  38.85
ntree=100  34.85  34.95  35.55  36.48  36.13  37.81  38.46
ntree=250  35.38  33.81  35.40  37.46  36.54  37.56  37.49
ntree=500  34.58  34.20  35.73  37.38  36.92  37.33  37.75
```

It seems that fewer trees results in a higher variance in error. It seems that fewer features per split also results in higher variance in the error, and as the number of features increases at first leads to lowere error, but then leads to increased error, likely due to overfitting.