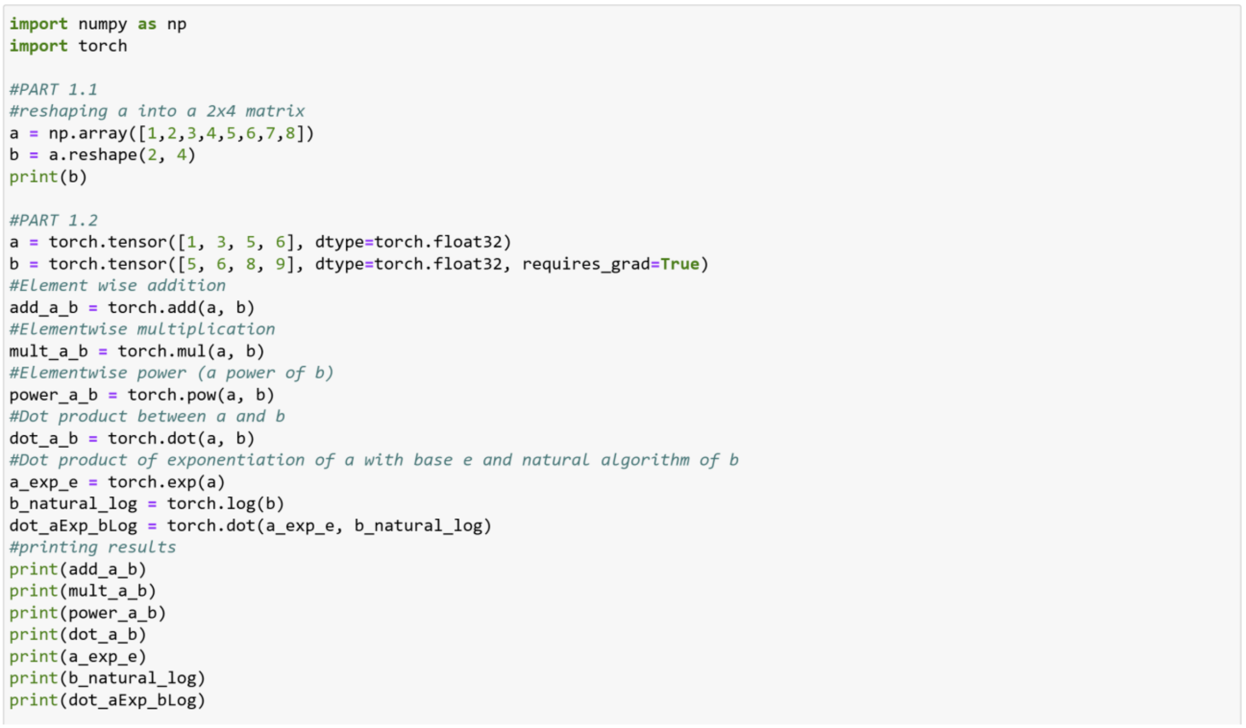
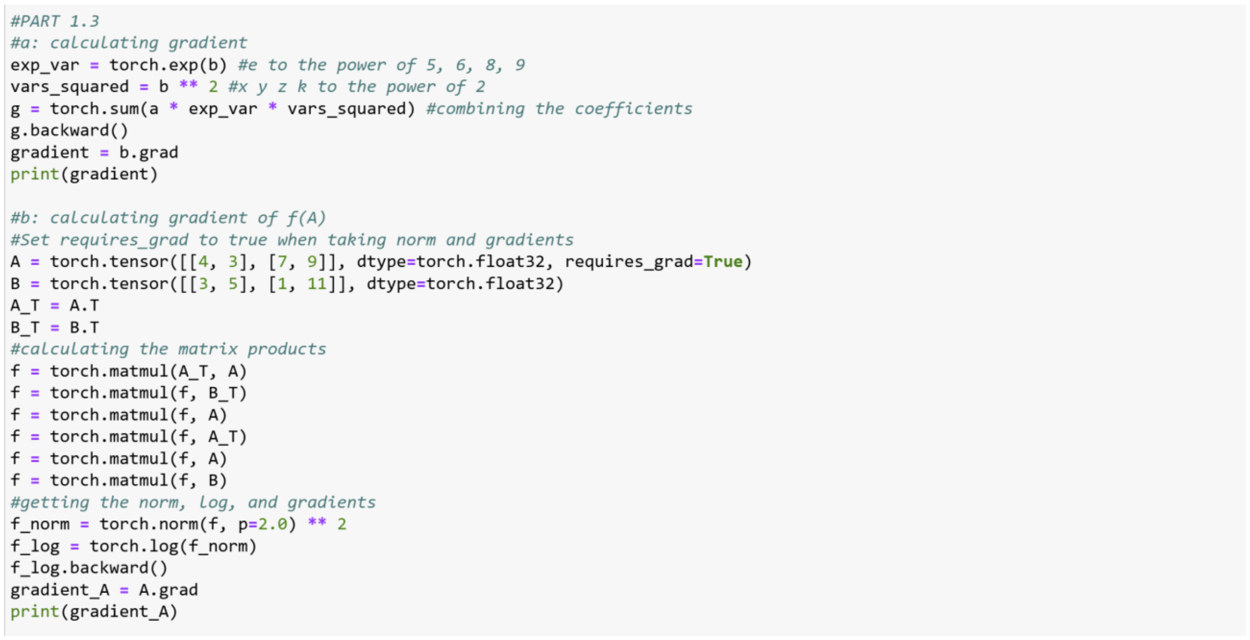
CS5785 Applied Machine Learning Homework 1 – Report

Please refer to my code for further information as it's commented.

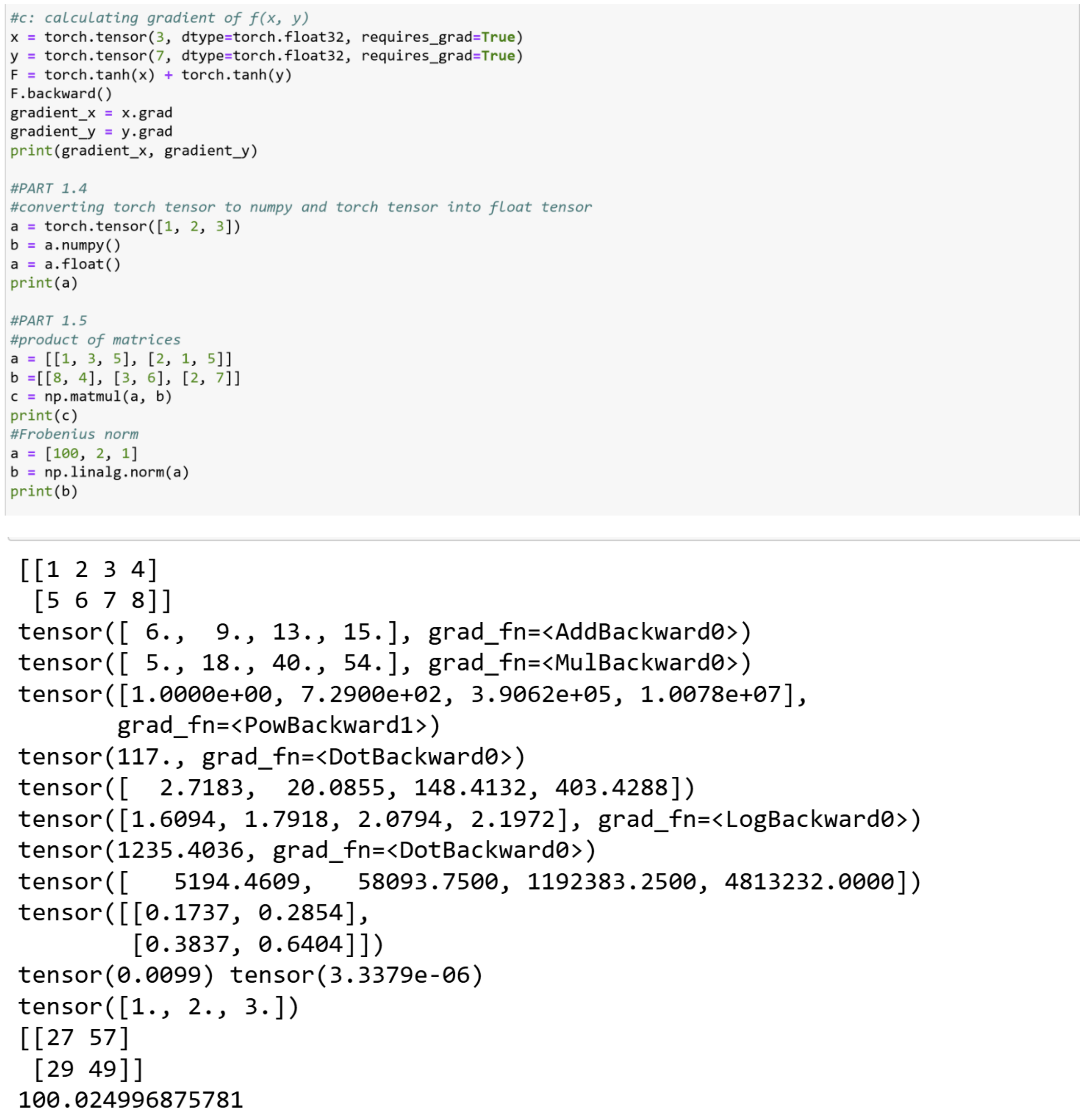
Warmup:



```python
import numpy as np
import torch

#PART 1.1
#reshaping a into a 2x4 matrix
a = np.array([1,2,3,4,5,6,7,8])
b = a.reshape(2, 4)
print(b)

#PART 1.2
a = torch.tensor([1, 3, 5, 6], dtype=torch.float32)
b = torch.tensor([5, 6, 8, 9], dtype=torch.float32, requires_grad=True)
#Element wise addition
add_a_b = torch.add(a, b)
#Elementwise multiplication
mult_a_b = torch.mul(a, b)
#Elementwise power (a power of b)
power_a_b = torch.pow(a, b)
#Dot product between a and b
dot_a_b = torch.dot(a, b)
#Dot product of exponentiation of a with base e and natural algorithm of b
a_exp_e = torch.exp(a)
b_natural_log = torch.log(b)
dot_aExp_bLog = torch.dot(a_exp_e, b_natural_log)
#printing results
print(add_a_b)
print(mult_a_b)
print(power_a_b)
print(dot_a_b)
print(a_exp_e)
print(b_natural_log)
print(dot_aExp_bLog)
```



```python
#PART 1.3
#a: calculating gradient
exp_var = torch.exp(b) #e to the power of 5, 6, 8, 9
vars_squared = b ** 2 #x y z k to the power of 2
g = torch.sum(a * exp_var * vars_squared) #combining the coefficients
g.backward()
gradient = b.grad
print(gradient)

#b: calculating gradient of f(A)
#Set requires_grad to true when taking norm and gradients
A = torch.tensor([[4, 3], [7, 9]], dtype=torch.float32, requires_grad=True)
B = torch.tensor([[3, 5], [1, 11]], dtype=torch.float32)
A_T = A.T
B_T = B.T
#calculating the matrix products
f = torch.matmul(A_T, A)
f = torch.matmul(f, B_T)
f = torch.matmul(f, A)
f = torch.matmul(f, A_T)
f = torch.matmul(f, A)
f = torch.matmul(f, B)
#getting the norm, log, and gradients
f_norm = torch.norm(f, p=2.0) ** 2
f_log = torch.log(f_norm)
f_log.backward()
gradient_A = A.grad
print(gradient_A)
```

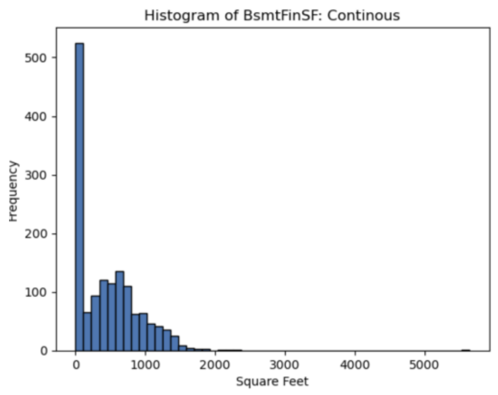


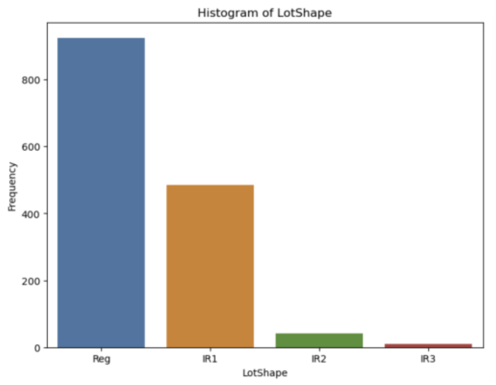
```
#c: calculating gradient of f(x, y)
x = torch.tensor(3, dtype=torch.float32, requires_grad=True)
y = torch.tensor(7, dtype=torch.float32, requires_grad=True)
F = torch.tanh(x) + torch.tanh(y)
F.backward()
gradient_x = x.grad
gradient_y = y.grad
print(gradient_x, gradient_y)

#PART 1.4
#converting torch tensor to numpy and torch tensor into float tensor
a = torch.tensor([1, 2, 3])
b = a.numpy()
a = a.float()
print(a)

#PART 1.5
#product of matrices
a = [[1, 3, 5], [2, 1, 5]]
b =[[8, 4], [3, 6], [2, 7]]
c = np.matmul(a, b)
print(c)
#Frobenius norm
a = [100, 2, 1]
b = np.linalg.norm(a)
print(b)
```

```
[[1 2 3 4]
 [5 6 7 8]]
tensor([ 6.,  9., 13., 15.], grad_fn=<AddBackward0>)
tensor([ 5., 18., 40., 54.], grad_fn=<MulBackward0>)
tensor([1.0000e+00, 7.2900e+02, 3.9062e+05, 1.0078e+07],
       grad_fn=<PowBackward1>)
tensor(117., grad_fn=<DotBackward0>)
tensor([  2.7183,  20.0855, 148.4132, 403.4288])
tensor([1.6094, 1.7918, 2.0794, 2.1972], grad_fn=<LogBackward0>)
tensor(1235.4036, grad_fn=<DotBackward0>)
tensor([   5194.4609,   58093.7500, 1192383.2500, 4813232.0000])
tensor([[0.1737, 0.2854],
        [0.3837, 0.6404]])
tensor(0.0099) tensor(3.3379e-06)
tensor([1., 2., 3.])
[[27 57]
 [29 49]]
100.024996875781
```

Kaggle:

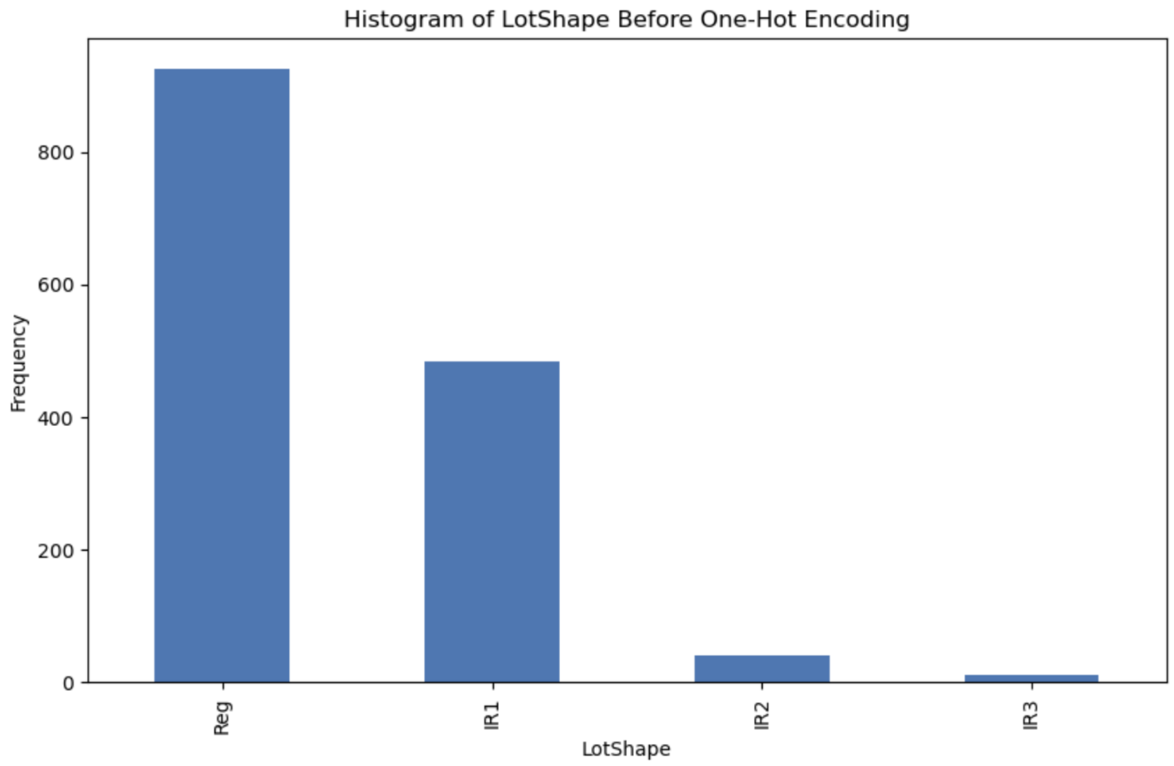
What I did for part two is downloading the training data and testing data into df2 and df1 respectively. Then I gave three examples of continuous data which would be BsmtFinSF, 1stFlrSF, and GarageArea since they are all represented by numerical data. Three examples of categorical data are LotShape, Neighborhood, and RoofStyle since these are categories that cannot be numbered (yet). I followed by plotting the histograms of BsmtFinSF and LotShape using matplotlib.

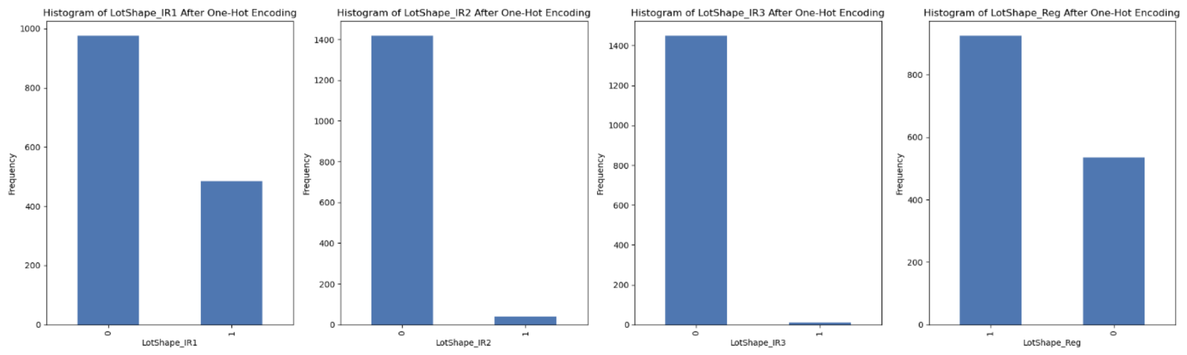




I cleaned the data of both testing and training data by replacing the NA values with 'None' because python can process those values over NA. I did the same thing with my numerical values. Preprocessing measures are important because otherwise, you will not be able to get accurate predictions and the code can potentially not be executable.

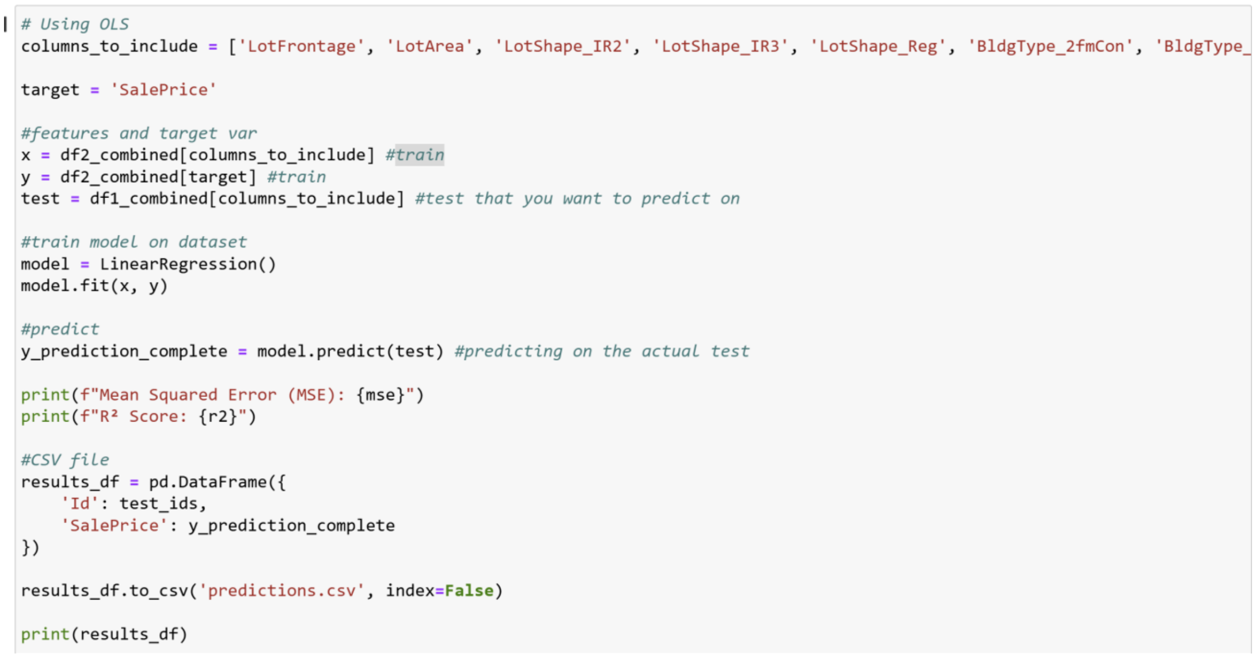
After cleaning the data, I one-hot encoded the entire data frames for both training and testing, and plotted the difference between LotShape before and after hot encoding. The reason why we do this is because by doing this, we can give non-numerical data (categorical) a way to process numerically and be able to run a regression.



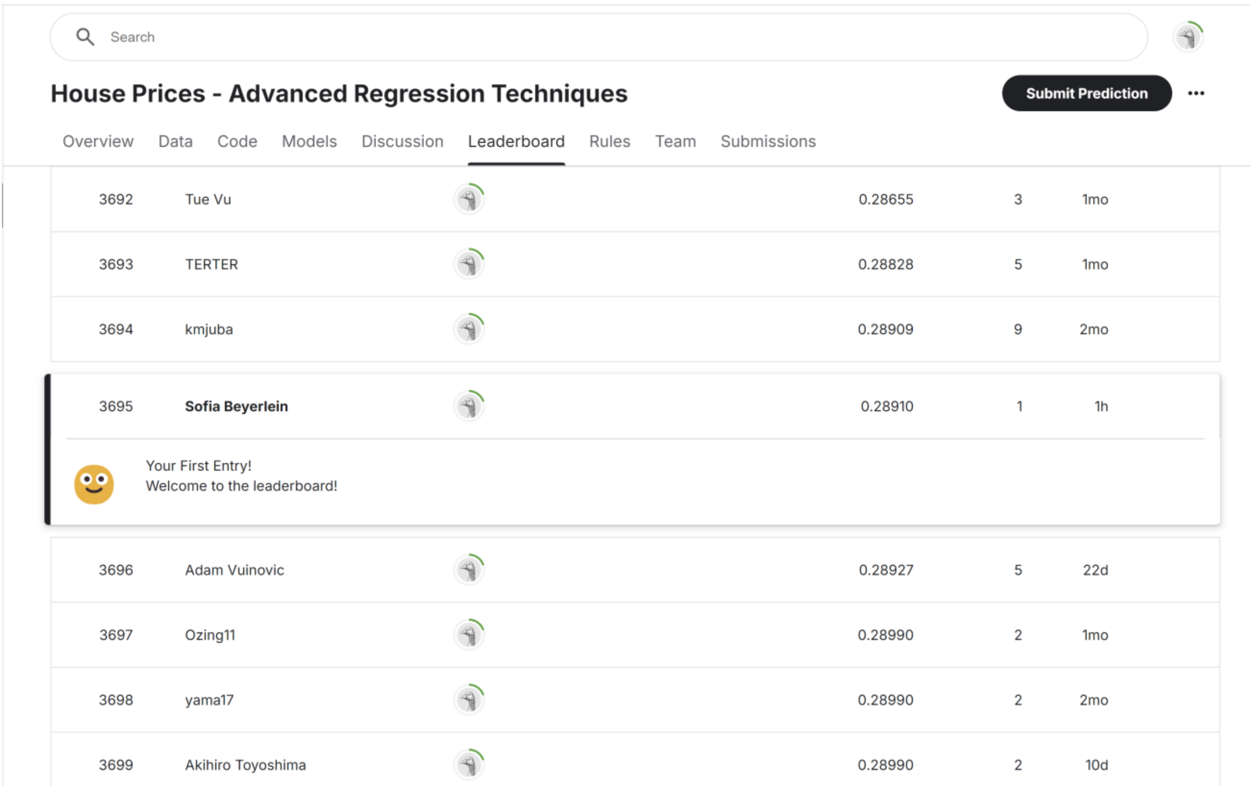


The last step is to train the model and predict the sales prices of the houses in the testing data frame. I started by including the columns columns_to_include = ['LotFrontage', 'LotArea', 'LotShape_IR2', 'LotShape_IR3', 'LotShape_Reg', 'BldgType_2fmCon', 'BldgType_Duplex', 'BldgType_Twnhs', 'BldgType_TwnhsE', 'HouseStyle_1.5Unf', 'HouseStyle_1Story', 'HouseStyle_2.5Unf', 'HouseStyle_2Story', 'HouseStyle_SFoyer', 'HouseStyle_SLvl', 'YearBuilt']. I chose these columns specifically because I believe lot frontage, lot area of the property, the property shape, the house style, and the year the house was built are decent indicators at what the price of the house would be.

I separated the y variable into a target = 'SalePrice'. I then created an x variable that included all the above columns of the training data frame y variable that only has ‘SalesPrice’ column of training data frame. I then created the test variable called test that has all of the columns_to_include for the testing data. I then made a Linear Regression model and then fit my (x, y) to that model. I then used that model to predict what the sales price of a home would be given the test variable. I calculated the MSE and R^2 values using sklearn metrics library and created a csv file that contained the id's and predicted SalesPrice.



```
# Using OLS
columns_to_include = ['LotFrontage', 'LotArea', 'LotShape_IR2', 'LotShape_IR3', 'LotShape_Reg', 'BldgType_2fmCon', 'BldgType_

target = 'SalePrice'

#features and target var
x = df2_combined[columns_to_include] #train
y = df2_combined[target] #train
test = df1_combined[columns_to_include] #test that you want to predict on

#train model on dataset
model = LinearRegression()
model.fit(x, y)

#predict
y_prediction_complete = model.predict(test) #predicting on the actual test

print(f"Mean Squared Error (MSE): {mse}")
print(f"R² Score: {r2}")

#CSV file
results_df = pd.DataFrame({
    'Id': test_ids,
    'SalePrice': y_prediction_complete
})

results_df.to_csv('predictions.csv', index=False)

print(results_df)
```



Search

House Prices - Advanced Regression Techniques

Submit Prediction

Overview Data Code Models Discussion Leaderboard Rules Team Submissions

3692	Tue Vu	0.28655	3	1mo
3693	TERTER	0.28828	5	1mo
3694	kmjuba	0.28909	9	2mo
3695	**Sofia Beyerlein**	0.28910	1	1h
3696	Adam Vuinovic	0.28927	5	22d
3697	Ozing11	0.28990	2	1mo
3698	yama17	0.28990	2	2mo
3699	Akihiro Toyoshima	0.28990	2	10d

Your First Entry!
Welcome to the leaderboard!

Written Portion

1. We must make assumptions in order for the machine to learn. If we make decisions only based from data alone, then the algorithm will fall into predictable behaviors rather than making "mistake" and learning from those mistakes.

2a) cost function = $J(\theta_0, \theta_1) = \sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2$

regression = $\hat{y}^{(i)} = \theta_0 + \theta_1 x^{(i)}$

Expand cost:

$J(\theta_0, \theta_1) = \sum_{i=1}^{n} (y^{(i)} - (\theta_0 + \theta_1 x^{(i)}))^2$

Calculate $\frac{\partial}{\partial\theta_0} J(\theta_0, \theta_1)$:

$J(\theta_0, \theta_1) = \sum_{i=1}^{n} (y^{(i)} - \theta_0 - \theta_1 x^{(i)})^2$

$$\begin{aligned}\frac{\partial}{\partial\theta_0} J(\theta_0, \theta_1) &= \sum_{i=1}^{n} 2(y^{(i)} - \theta_0 - \theta_1 x^{(i)}) \cdot \frac{\partial}{\partial\theta_0}(y^{(i)} - \theta_0 - \theta_1 x^{(i)}) \\ &= \sum_{i=1}^{n} 2(y^{(i)} - \theta_0 - \theta_1 x^{(i)}) \cdot -1 \\ &= -2\sum_{i=1}^{n} (y^{(i)} - \theta_0 - \theta_1 x^{(i)})\end{aligned}$$

$$\begin{aligned}\frac{\partial}{\partial\theta_1} J(\theta_0, \theta_1) &= \sum_{i=1}^{n} (y^{(i)} - (\theta_0 + \theta_1 x^{(i)}))^2 \\ &= \sum_{i=1}^{n} 2(y^{(i)} - \theta_0 - \theta_1 x^{(i)}) \cdot \frac{\partial}{\partial\theta_1}(y^{(i)} - \theta_0 - \theta_1 x^{(i)}) \\ &= \sum_{i=1}^{n} 2(y^{(i)} - \theta_0 - \theta_1 x^{(i)}) \cdot (-x^{(i)}) \\ &= -2\sum_{i=1}^{n} x^{(i)}(y^{(i)} - \theta_0 - \theta x_1^{(i)})\end{aligned}$$

2b) $\frac{\partial}{\partial \theta_0} J(\theta_0^*, \theta_1) = 0$

$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1^*) = 0$

Prove the following : $\theta_0^* = \bar{y} - \theta_1 \bar{x}$ and $\theta_1^* = \frac{\sum_{i=1}^n x^{(i)}(y^{(i)} - \bar{y})}{\sum_{i=1}^n x^{(i)}(x^{(i)} - \bar{x})}$

Note : $\bar{x} = \frac{1}{n} \sum_{i=1}^n x^{(i)}$ and $\bar{y} = \frac{1}{n} \sum_{i=1}^n y^{(i)}$

$$\frac{\partial J(\theta_0, \theta_1)}{\partial \theta_0} = -2 \sum_{i=1}^n (y^{(i)} - \theta_0 - \theta_1 x^{(i)}) = 0$$

$$\sum_{i=1}^n (y^{(i)} - \theta_0 - \theta_1 x^{(i)}) = 0$$

$$\sum_{i=1}^n y^{(i)} - \sum_{i=1}^n \theta_0 - \sum_{i=1}^n \theta_1 x^{(i)} = 0$$

$$\frac{\sum_{i=1}^n y^{(i)} - n\theta_0 - \theta_1 \sum_{i=1}^n x^{(i)} = 0}{n}$$

substitute: $$\frac{1}{n} \sum_{i=1}^n y^{(i)} - \theta_0 - \frac{1}{n} \theta_1 \sum_{i=1}^n x^{(i)} = 0$$

$$\bar{y} - \theta_0 - \theta_1 \bar{x} = 0$$

$$\theta_0^* = \bar{y} - \theta_1 \bar{x}$$

$$\frac{\partial J(\theta_0, \theta_1)}{\partial \theta_1} = -2 \sum_{i=1}^n x^{(i)} (y^{(i)} - \theta_0 - \theta x_1^{(i)})$$

$$= -2 \sum_{i=1}^n x^{(i)} (y^{(i)} - \theta_0 - \theta x_1^{(i)}) = 0$$

$$\sum_{i=1}^n x^{(i)} (y^{(i)} - \theta_0 - \theta x_1^{(i)}) = 0$$

$$x^{(i)} \sum_{i=1}^n (y^{(i)} - \theta_0 - \theta x_1^{(i)}) = 0$$

$$\sum_{i=n}^n x^{(i)} y^{(i)} - \sum_{i=1}^n x^{(i)} \theta_0 - \sum_{i=1}^n - \theta_1 (x^{(i)})^2 = 0$$

$$\sum_{i=1}^n x^{(i)} y^{(i)} - \theta_0 \sum_{i=1}^n x^{(i)} - \theta_1 \sum_{i=1}^n x^{(i)} x^{(i)} = 0$$

substitute $$\sum_{i=1}^n x^{(i)} y^{(i)} - (\bar{y} - \theta_1 \bar{x}) \sum_{i=1}^n x^{(i)} - \theta_1 \sum_{i=1}^n x^{(i)} x^{(i)} = 0$$

$$\sum_{i=1}^n x^{(i)} y^{(i)} - \bar{y} \sum_{i=1}^n x^{(i)} + \theta_1 \bar{x} \sum_{i=1}^n x^{(i)} - \theta_1 \sum_{i=1}^n x^{(i)} x^{(i)} = 0$$

$$\sum_{i=1}^n x^{(i)} y^{(i)} - \bar{y} \sum_{i=1}^n x^{(i)} + \theta_1 (\bar{x} \sum_{i=1}^n x^{(i)} - \sum_{i=1}^n x^{(i)} x^{(i)}) = 0$$

substitute $$\sum_{i=0}^n x^{(i)} y^{(i)} - n\bar{y}\bar{x} = \theta_1 (\sum_{i=1}^n (x^{(i)})^2 - n\bar{x}^2)$$

solve for θ_1 $$\theta_1 = \frac{\sum_{i=0}^n (x^{(i)} - \bar{x})(y^{(i)} - \bar{y})}{\sum_{i=0}^n (x^{(i)} - \bar{x})^2}$$

$$\theta_1^* = \frac{\sum_{i=1}^n (x^{(i)} - \bar{x})(y^{(i)} - \bar{y})}{\sum_{i=1}^n x^{(i)} (x^{(i)} - \bar{x})}$$

2c) Calculate the residuals

$$\sum_{i=1}^{n} e^{i} = \sum_{i=1}^{n} (y^{i} - (\theta_0^{*} + \theta_1^{*} x^{i}))$$

$$\sum_{i=1}^{n} e^{i} = \sum_{i=1}^{n} (y^{i} - \theta_0^{*} - \theta_1^{*} x^{i})$$

$$\sum_{i=1}^{n} e^{i} = \sum_{i=1}^{n} (y^{i} - (\bar{y} - \theta_1^{*} \bar{x}) - \theta_1^{*} x^{i})$$

$$\sum_{i=1}^{n} e^{i} = \sum_{i=1}^{n} (y^{i} - \bar{y} + \theta_1^{*} \bar{x} - \theta_1^{*} x^{i})$$

$$\sum_{i=1}^{n} e^{i} = \sum_{i=1}^{n} y^{i} - \sum_{i=1}^{n} \bar{y} + \theta_1^{*} \sum_{i=1}^{n} \bar{x} - \theta_1^{*} \sum_{i=1}^{n} x^{i} \qquad \text{substitute}$$

$$\sum_{i=1}^{n} e^{i} = \cancel{n\bar{y}} - \cancel{n\bar{y}} + \cancel{\theta_1^{*} n\bar{x}} - \cancel{\theta_1^{*} n\bar{x}}$$

$$\sum_{i=1}^{n} e^{i} = 0 \qquad \text{simplify}$$

We learn that the residuals cancel out and that the predicted values are equal to the actual values that the regression line gave.

3) What could've gone wrong is that the data in which the models had a lot of outliers, and/or there was not enough data to train on. Two potential explanations is noisy data causes the model to not learn the patterns and the quality and quantity of data impacts the performance of a model significantly.

4a) i. We should remove variables that are collinear if we care about model interpretability because we could not say with certainty which variable is the one causing changes on the dependent variable.

ii. We should keep all variables if we want to make the best prediction because the model will have all the data to make a prediction with and making the model have better accuracy.

4b) i. The assumption is invalidated because changing the feature (temperature) should give one outcome instead of two. In this case it does not follow monotonicity and would not be a linear relationship.

ii. A potential solution is is to turn the function into a quadratic function.

4c) i. The assumption that is invalidated because the variables are not independent and have effects in each other.

ii. A potential solution is to use one-hot encoding and to see of either variable is present.