Bhandari_Nischal_in_Class_assignment

September 26, 2023

[1]:
```python
import pandas as pd

wire_df = pd.read_excel('wire_bond.xlsx', index_col=0, header=0)
wire_df.head()
```

[1]:
```
                           y           x1           x2           x3  \
observation
NaN          pull strength   die height  post height  loop height
1.0                      8          5.2           17         28.6
2.0                      8          5.2         19.6         29.6
3.0                    8.3          5.8         19.8         32.4
4.0                    8.5          6.4         19.6           31

                       x4                     x5                      x6
observation
NaN          wire lenghth  bond width on the die  bond width on the post
1.0                    83                    1.9                     1.6
2.0                  94.9                    2.1                     2.3
3.0                  89.7                    2.1                     1.8
4.0                  96.2                      2                       2
```

[2]:
```python
# removing the first row
wire_df = wire_df.iloc[1:]

# renaming the columns
wire_df.columns = ['y', 'x1', 'x2', 'x3', 'x4', 'x5', 'x6']
# remove first column and set index default
wire_df = wire_df.iloc[:, 0:].reset_index(drop=True)
wire_df.head()
```

[2]:
```
     y   x1    x2    x3    x4   x5   x6
0    8  5.2    17  28.6    83  1.9  1.6
1    8  5.2  19.6  29.6  94.9  2.1  2.3
2  8.3  5.8  19.8  32.4  89.7  2.1  1.8
3  8.5  6.4  19.6    31  96.2    2    2
4  8.8  5.8  19.4  32.4  95.6  2.2  2.1
```

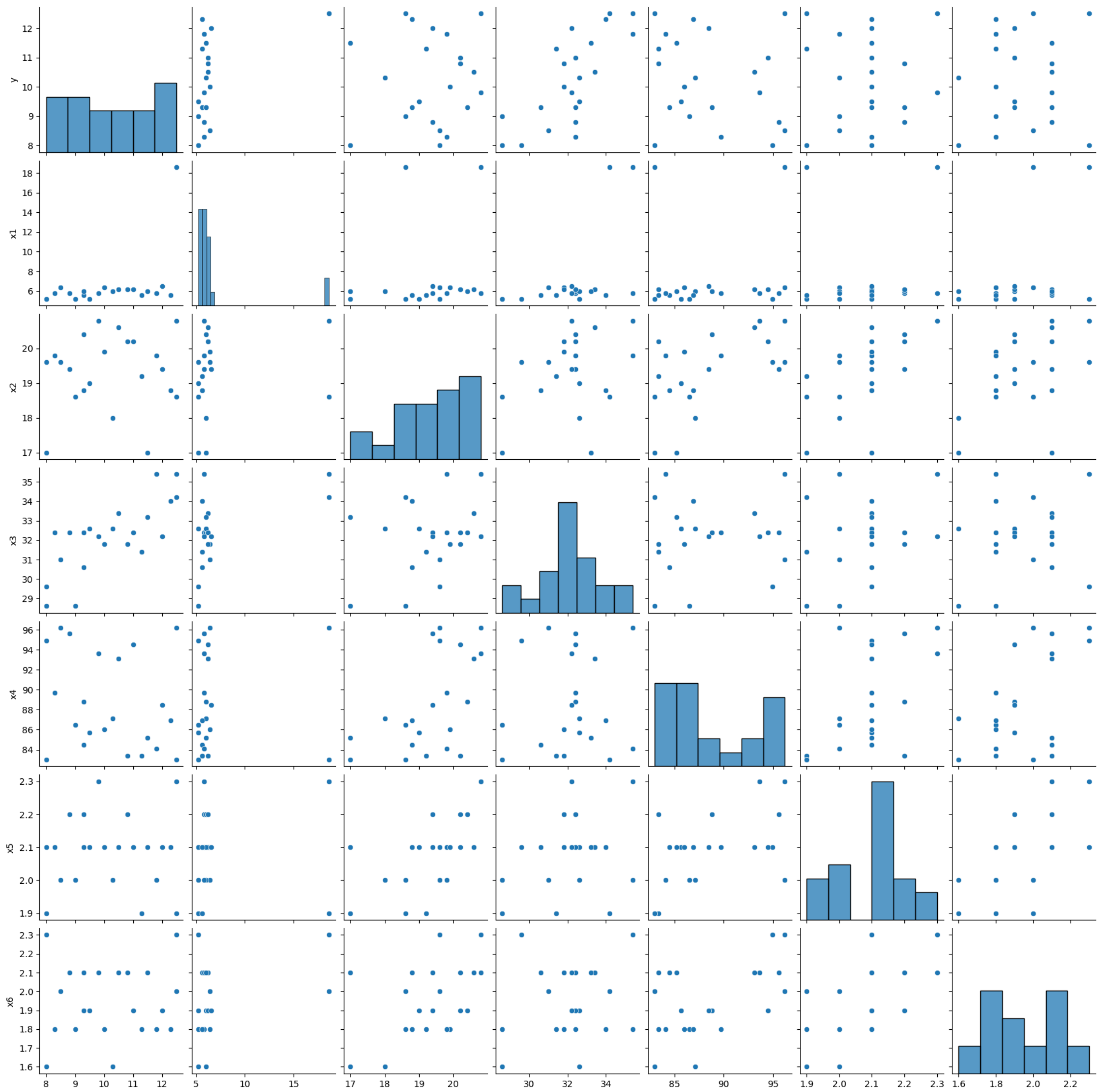
[3]:
```
wire_df.describe()
```

[3]:
```
           y    x1    x2    x3    x4    x5    x6
count   22.0  22.0  22.0  22.0  22.0  22.0  22.0
unique  19.0   8.0  14.0  14.0  19.0   5.0   6.0
top      8.0   5.2  17.0  32.4  83.0   2.1   1.8
freq     2.0   4.0   2.0   4.0   2.0  10.0   6.0
```

[4]:
```
# checking for missing values
wire_df.isna().sum()
```

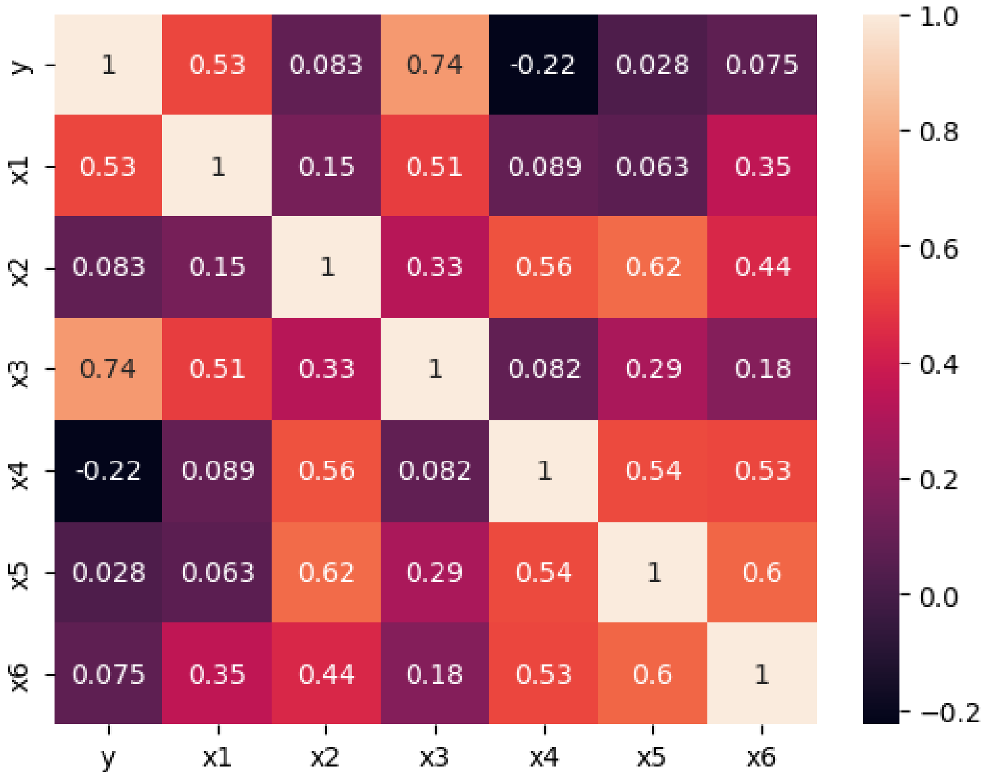
[4]:
```
y     0
x1    0
x2    0
x3    0
x4    0
x5    0
x6    0
dtype: int64
```

[5]:
```
# data analysis
import seaborn as sns
import matplotlib.pyplot as plt
sns.pairplot(wire_df)
plt.show()
```



pull strength seems to be positively correlated with die height (x1) and bond width on the post (x6)

[21]:
```
# CORRELATION MATRIX
import seaborn as sns
corrMatrix = wire_df.corr()
sns.heatmap(corrMatrix, annot=True)
plt.show()
```



yeah, i was right above. only to the extent though. x1 has strong r coefficient than x6. x3 has the strongest correlation of all. let's see how the model places weights in these features.

0.0.1 Trying Linear Regression First

[8]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics


# Define the dependent variable (y) and independent variables (X)
X = wire_df[['x1', 'x2', 'x3', 'x4', 'x5', 'x6']]
y = wire_df['y']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)

# Train the model
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

```python
# Make predictions using the testing set
y_pred = regressor.predict(X_test)

# Evaluate the model
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,␣
↪y_pred)))
```

```
Mean Absolute Error: 0.7758713710719516
Mean Squared Error: 1.1796743010856483
Root Mean Squared Error: 1.086128123697038
```

[9]:

```python
from sklearn.metrics import r2_score
r2_score = r2_score(y_true=y_test,y_pred=y_pred)
print(r2_score)
```

```
0.455467918627378
```

The amount of variation explained is 45 %. So only 45% of the variation in the change in strength is explained by the other variables if linear regression method is used.

0.0.2 Next: Ordinary Least Square

[10]:

```python
import statsmodels.formula.api as smf
wire_df[['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'y']] = wire_df[['x1', 'x2', 'x3',␣
↪'x4', 'x5', 'x6', 'y']].apply(pd.to_numeric, errors='coerce')
lm_mul = smf.ols(formula='y ~ x1 + x2 + x3 + x4 + x5 + x6', data=wire_df).fit()
print(lm_mul.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.677
Model:                            OLS   Adj. R-squared:                  0.548
Method:                 Least Squares   F-statistic:                     5.244
Date:                Tue, 26 Sep 2023   Prob (F-statistic):            0.00430
Time:                        15:29:46   Log-Likelihood:                -27.051
No. Observations:                  22   AIC:                             68.10
Df Residuals:                      15   BIC:                             75.74
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.6737      5.891      0.114      0.910     -11.883      13.231
x1             0.0723      0.076      0.956      0.354      -0.089       0.233
x2             0.0226      0.286      0.079      0.938      -0.587       0.633
```

```
x3             0.5602      0.156      3.601      0.003       0.229       0.892
x4            -0.0978      0.061     -1.592      0.132      -0.229       0.033
x5            -1.0217      3.021     -0.338      0.740      -7.461       5.417
x6             0.7078      1.664      0.425      0.677      -2.839       4.255
==============================================================================
Omnibus:                        0.490   Durbin-Watson:                   1.438
Prob(Omnibus):                  0.783   Jarque-Bera (JB):                0.288
Skew:                           0.265   Prob(JB):                        0.866
Kurtosis:                       2.814   Cond. No.                     2.67e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 2.67e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In general, multiple linear regression would be applicable for this data since the data types in each columns is numerical. If we choose different models, OLS performs better because it accounts to ~68% in the variability of pull strength based on all the features used compared to the linear regression model which only accounted for ~45%.

The p-values for all features are greater than alpha value of 0.05 except x3 where alpha value where it is 0.03. So we reject the null hypothesis in case of x3.

In the hindsight, the correlation matrix also show a strong, positive correlation between y and x3.

The R-squared value for OLS model is 0.677. So it explains ~68% of variability.

[11]:
```python
# 95% Confidence interval for each of the  j's in your model
confidence_interval = lm_mul.conf_int(alpha=0.05)
print(' 95 % Confidence Interval:\n', confidence_interval)
```

```
 95 % Confidence Interval:
                    0          1
Intercept -11.883244  13.230595
x1         -0.088842   0.233431
x2         -0.587462   0.632592
x3          0.228599   0.891846
x4         -0.228725   0.033129
x5         -7.460726   5.417317
x6         -2.839084   4.254750
```

Interpretation of Confidence Interval of Model Coefficients β_0 and β_1 The 95% confidence interval for β_0 is [-11.883244, 13.230595] and the 95% confidence interval for other features varies.

We can conclude that in the absence of any other affects, pull strength of a wire, on average, fall somewhere between -11.883244 and 13.230595 units.

Other features have their own magnitude of effects. For example, one with the most positive effect on the pull strength of a wire, pull strength increases from 22 to 89 units in general when loop height is increased by 100 units.

[12]:
```python
# just looking at the slope of x4 in multiple linear regression
slope_unit4 = lm_mul.params['x4']
print('Slope of wire length 4:', slope_unit4)
```

```
Slope of wire length 4: -0.09779833603224988
```

[13]:
```python
# printing all weights
lm_mul.params
```

[13]:
```
Intercept    0.673675
x1           0.072295
x2           0.022565
x3           0.560223
x4          -0.097798
x5          -1.021704
x6           0.707833
dtype: float64
```

holding all features fixed, a unit change in x4 decresease the average value of y by 0.098 iin multiple linear regression.

d) Holding all else fixed, how does a unit change in x4 change the average value of y? ### Simple Linear Regression As SLS ignores the effect of other five varaibles and focus on mostly x4, let's try it.

[14]:
```python
import statsmodels.formula.api as smf

# 3. Train the model
lm = smf.ols(formula='y ~ x4', data=wire_df).fit()

# coefficients of the trained model

lm.params
```

[14]:
```
Intercept    16.465740
x4           -0.070386
dtype: float64
```

Interpretation of Model Coefficients β_0 and β_1

$$y = \beta_0 + \beta_1 x = 16.465740 - 0.070386 \times x4$$

$\beta_1 = 0.070386$: An additional 1000 on wire length is associated with decreasing of pulling strength by approximately 704 of the wire. Note: excluding other features increase the effect of wirelength (from -0.098 to -0.070)

e. For a specimen with x1 = 5.5, x2 = 19.3, x3 = 30.2, x4 = 90, x5 = 2, and x6 = 1.85 find the predicted value of y.

0.0.3 Fitting a Multiple Linear Regression model

$Y = \beta_0 + \beta_1 X_1 + ... + \beta_n X_n$

[15]:
```
lm_mul = smf.ols(formula='y ~ x1 + x2 + x3 + x4 + x5 + x6', data=wire_df).fit()
```

[16]:
```
# testing the new data
new_data = {'x1': [5.5], 'x2': [19.3], 'x3': [30.2], 'x4': [90], 'x5': [2],␣
 ↪'x6': [1.85]}
new_df = pd.DataFrame(new_data)
predicted_y = lm_mul.predict(new_df)
print("The predicted value of y for the given specimen is: ", predicted_y[0])
```

```
The predicted value of y for the given specimen is:  8.889750471014537
```

0.0.4 Checking Model Assumptions

[17]:
```
# take one data from the dataframe and check the model performance
import random
random_index = random.randint(0, len(wire_df)-1)
random_data = wire_df.iloc[random_index]
predicted_y = lm_mul.predict(random_data[['x1', 'x2', 'x3', 'x4', 'x5', 'x6']].
 ↪to_frame().T)
print("Predicted value of y for the randomly selected data is: ", predicted_y)
print(random_data)
```

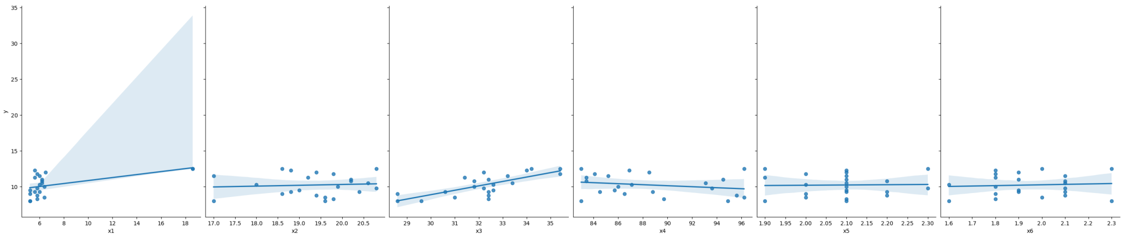
```
Predicted value of y for the randomly selected data is:  6    9.722465
dtype: float64
y      9.3
x1     5.6
x2    18.8
x3    30.6
x4    84.5
x5     2.1
x6     2.1
Name: 6, dtype: float64
```

The predicted value is 9.722465 where as the actual value is 9.3, so the model is not super reliable but not extremely worse either.

0.0.5 Checking model assumptions

[18]:
```
# linearity
import seaborn as sns
import matplotlib.pyplot as plt
```

```python
sns.pairplot(data = wire_df, x_vars=['x1', 'x2', 'x3', 'x4', 'x5', 'x6'],
    y_vars=['y'],height=6, aspect=0.8, kind='reg')
plt.show()
```



[19]:
```python
# Mean of residuals for linear regression
residuals = y.values.reshape(-1,1)-y_pred
mean_residuals = np.mean(residuals)
print("Mean of Residuals for linear regression {}".format(mean_residuals))
```

```
Mean of Residuals for linear regression -0.25139825017226475
```

[20]:
```python
# Checking mean of residuals for OLS
mean_lm_mul_resid = np.mean(lm_mul.resid)
print("Mean of Residuals for OLS method {}".format(mean_lm_mul_resid))
```

```
Mean of Residuals for OLS method 8.881784197001252e-16
```

the assumptions of linear regression that the mean of the residuals should be zero is not validated by our models. But OLS performs much better and the mean difference is close to zero.

[ ]:
```python

```