Conceptual

Table of contents

Import notebook funcs

```
from notebookfuncs import *
```

Import user funcs

```
from userfuncs import *
```

Import libraries

```
from sympy import symbols, solve
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
```

```python
import statsmodels.api as sm
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import pandas as pd
```

1. Describe the null hypotheses to which the p-values given in Table 3.4 correspond. Explain what conclusions you can draw based on these p-values. Your explanation should be phrased in terms of sales, TV, radio, and newspaper, rather than in terms of the coefficients of the linear model.

	Coefficient	Standard Error	t-statistic	p-value
Intercept	2.939	0.3119	9.12	< 0.0001
TV	0.046	0.0014	32.81	< 0.0001
Radio	0.189	0.0086	21.89	< 0.0001
Newspaper	-0.001	0.0059	-0.18	0.8599

- The null hypotheses to which the given p-values correspond are:
- $H_0 : \beta_{Intercept} = \beta_{TV} = \beta_{Radio} = \beta_{Newspaper} = 0$
- The p-values for Intercept, TV and Radio are significant i.e., less that 0.01.
- Hence, we can reject the null hypotheses that $\beta_{Intercept}$, β_{TV} and β_{Radio} are zero i.e., the intercept and coefficient values for TV and Radio are significant in the multilinear regression model.
- The model thus becomes $Sales = 2.939 + 0.0046 * TV + 0.189 * Radio$
- The Intercept value implies that in the absence of any advertsing spend on TV and Radio, the sales would on average be $2.939 * 1000 = 2939$ units.
- The coefficient of 0.046 on TV suggests that for every \$1000 spent on TV advertising, the sales units increase by 0.0046 * 1000 = 46 units. Radio spend remaining constant.
- Similarly, for every additional 1000 dollars spent on radio, the sales units increase by 0.189 * 1000 = 189 units keeping TV spending constant.

2. Carefully explain the differences between the KNN classifier and KNN regression methods.

KNN Classifier

The KNN Classifier deals in probabilities and selects the likeliest or most frequent estimator of the category (qualitative variable) from the nearest k neighbours. It selects the category with the highest probability from all the k-nearest neighbours of the data point chosen. The co-domain is a discrete space.

KNN Regression

The KNN Regression, on the other hand, usually selects the average of the k nearest neighbours of the data point. You could also use the median or weighted average value of the k nearest neighbours. The co-domain is a continuous space.

Similarities:

1. Both use proximity-based approach
2. Depend on feature similarity
3. Use K-nearest neighbors to make predictions
4. No explicit model training

Differences:

Classification:

1. Predicts class labels (categorical)
2. Output: Class label (e.g., 0/1, yes/no)
3. Distance metric: Typically Euclidean, Hamming, or Minkowski
4. Decision boundary: Non-linear, based on KNN
5. Evaluation metrics: Accuracy, Precision, Recall, F1-score

Regression:

1. Predicts continuous values (numerical)
2. Output: Continuous value (e.g., price, temperature)
3. Distance metric: Typically Euclidean or Minkowski
4. Decision boundary: Non-linear, based on KNN
5. Evaluation metrics: MSE, MAE, R-squared, RMSE

Key differences:

1. Output type (categorical vs. numerical)
2. Distance metric suitability
3. Evaluation metrics

KNN Classification:

1. Majority voting (most common class label)
2. Weighted voting (distance-weighted class labels)

KNN Regression:

1. Average neighboring values (simple average)
2. Weighted average (distance-weighted average)
3. Median of neighboring values (median)

Hyperparameters:

1. K (number of nearest neighbors)
2. Distance metric
3. Weighting scheme (uniform or distance-based)

Advantages:

1. Simple implementation
2. No explicit model training
3. Handles non-linear relationships

Disadvantages:

1. Computationally expensive
2. Sensitive to noise and outliers
3. Choice of K and distance metric

Real-world applications:

Classification:

- Image classification
- Text categorization
- Spam detection

Regression:

- Predicting house prices
- Energy consumption forecasting
- Stock price prediction

Here are the equations and explanations for KNN Classification and Regression:

KNN Classification

Majority Voting

$$y = argmax \sum_{i=1}^{K} I(y_i = c)$$

where:

- y: predicted class label
- K: number of nearest neighbors
- y_i: class label of i_{th} nearest neighbor
- c: class label

- $I()$: indicator function (1 if true, 0 otherwise)

Weighted Voting

$$y = argmax \sum_{i=1}^{K} w_i I(y_i = c)$$

where:

- w_i: weight assigned to ith nearest neighbor (typically 1/distance)

KNN Regression

Simple Average

$$y = (1/K) \sum_{i=1}^{K} y_i$$

where:

- y: predicted value
- K: number of nearest neighbors
- y_i: value of ith nearest neighbor

Weighted Average

$$y = \left(\sum_{i=1}^{K} w_i y_i \right) / \left(\sum_{i=1}^{K} w_i \right)$$

where:

- w_i: weight assigned to ith nearest neighbor (typically 1/distance)

Distance Metrics

- **Euclidean distance**: $\sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$
- **Minkowski distance**: $\sqrt[p]{\sum_{i=1}^{n} |xi - yi|^p}$
- **Hamming distance**: $\sum_{i=1}^{n} I(x_i \neq y_i)$

KNN Algorithm

1. Choose K and distance metric.
2. Calculate distances between query point and training points.
3. Select K nearest neighbors.
4. Predict class label (classification) or value (regression).

References: 1. https://stats.stackexchange.com/questions/364351/regression-knn-model-vs-classification-knn-model 2. https://stackoverflow.com/questions/64990030/difference-between-classification-and-regression-in-k-nearest-neighbor 3. MetaAI

3. Suppose we have a data set with five predictors, $X__1 = GPA, X__2 = IQ, X__3 = Level $ (1 for College and 0 for High School), $X_4 = Interaction$ between GPA and IQ, and $X_5 = Interaction$ between GPA and Level. The response is starting salary after graduation (in thousands of dollars). Suppose we use least squares to fit the model, and get $\beta_0 = 50, \beta_1 = 20, \beta_2 = 0.07, \beta_3 = 35, \beta_4 = 0.01, \beta_5 = -10$.

(a) Which answer is correct, and why?

i. For a fixed value of IQ and GPA, high school graduates earn more, on average, than college graduates.

ii. For a fixed value of IQ and GPA, college graduates earn more, on average, than high school graduates.

iii. For a fixed value of IQ and GPA, high school graduates earn more, on average, than college graduates provided that the GPA is high enough.

iv. For a fixed value of IQ and GPA, college graduates earn more, on average, than high school graduates provided that the GPA is high enough.

```
gpa, iq, level , gpa_iq, gpa_level  = symbols("gpa iq level gpa.iq gpa.level")
equation = 50  + 20 * gpa + 0.07 * iq + 35 * level + 0.01 * gpa_iq - 10 * gpa_level
```

$20gpa + 0.01gpa.iq - 10gpa.level + 0.07iq + 35level + 50$

```
eqn_highschool = equation.subs([(level,0), (gpa_level, 0 )])
```

$20gpa + 0.01gpa.iq + 0.07iq + 50$

```
eqn_college = equation.subs([(level,1), (gpa_level, gpa)])
```

$10gpa + 0.01gpa.iq + 0.07iq + 85$

```
diff = (eqn_college - eqn_highschool) > 0
```

$35 - 10gpa > 0$

```
solve(diff)
```

$$-\infty < gpa \wedge gpa < \frac{7}{2}$$

From the data above: - We can conclude that college graduates earn more than high school graduates for gpa values less than 3.5. - For gpa values greater than or equal to 3.5, high school graduates earn more than college graduates on average which is equivalent to point (iii), provided the gpa is high enough.

(b) Predict the salary of a college graduate with IQ of 110 and a GPA of 4.0.

```
equation.subs([(gpa, 4.0), (iq,110), (level, 1), (gpa_iq, 110*4.0), (gpa_level, 4.0
 ↪  * 1)]) * 1000
```

```
137100.0
```

(c) True or false: Since the coefficient for the GPA/IQ interaction term is very small, there is very little evidence of an interaction effect. Justify your answer.

- The significance of an effect irrespective of whether it's an interaction or not depends of the p-value of its coefficient and not on the coeffcient value.
- Unless you've standardized the variables, the scale of each variable may differ from each other hugely and even a small coefficient can have a significant effect on the response if the variable values are quite large.
- You might also wish to look up Lasso and Ridge regression models to discover two different types of regressions where coefficients are either shrunk close to zero or omitted from the model to enhance interpretability.

4. I collect a set of data (n = 100 observations) containing a single predictor and a quantitative response. I then fit a linear regression model to the data, as well as a separate cubic regression, i.e. $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \epsilon$.

(a) Suppose that the true relationship between X and Y is linear, i.e. $Y = \beta_0 + \beta_1 X + \epsilon$. Consider the training residual sum of squares (RSS) for the linear regression, and also the training RSS for the cubic regression. Would we expect one to be lower than the other, would we expect them to be the same, or is there not enough information to tell? Justify your answer.

Given that the true relationship is linear, it is quite likely that the training RSS for the cubic regression tends to overfit the data and thus the training RSS for the cubic regression is lower than the one for the linear regression.

```
# Generate data
np.random.seed(0)
```

```
x = np.random.uniform(-10, 10, 100)
y = 2 * x + 1 + np.random.normal(0, 1, 100)

# Linear Regression
lr = LinearRegression()
lr.fit(x.reshape(-1, 1), y)
y_pred_lr = lr.predict(x.reshape(-1, 1))

# Cubic Regression
poly = PolynomialFeatures(degree=3, include_bias=False)
x_poly = poly.fit_transform(x.reshape(-1, 1))
lr_poly = LinearRegression()
lr_poly.fit(x_poly, y)
y_pred_poly = lr_poly.predict(x_poly)

# Print coefficients and RSS
print("Linear Regression: Coefficients and Intercept")
print(lr.coef_, lr.intercept_)
print("RSS:", np.sum((y - y_pred_lr) ** 2))

x2 = sm.add_constant(x)
est = sm.OLS(y, x2)
est2 = est.fit()
print(get_results_df(est2))

print("\nCubic Regression: Coefficients and Intercept")
print(lr_poly.coef_, lr_poly.intercept_)
print("RSS:", np.sum((y - y_pred_poly) ** 2))

x2 = sm.add_constant(x_poly)
est = sm.OLS(y, x2)
est2 = est.fit()
print(get_results_df(est2))

# Plot data and predictions
plt.scatter(x, y)
plt.plot(x, y_pred_lr, label="Linear")
plt.plot(x, y_pred_poly, label="Cubic")
plt.legend()
plt.show()
```

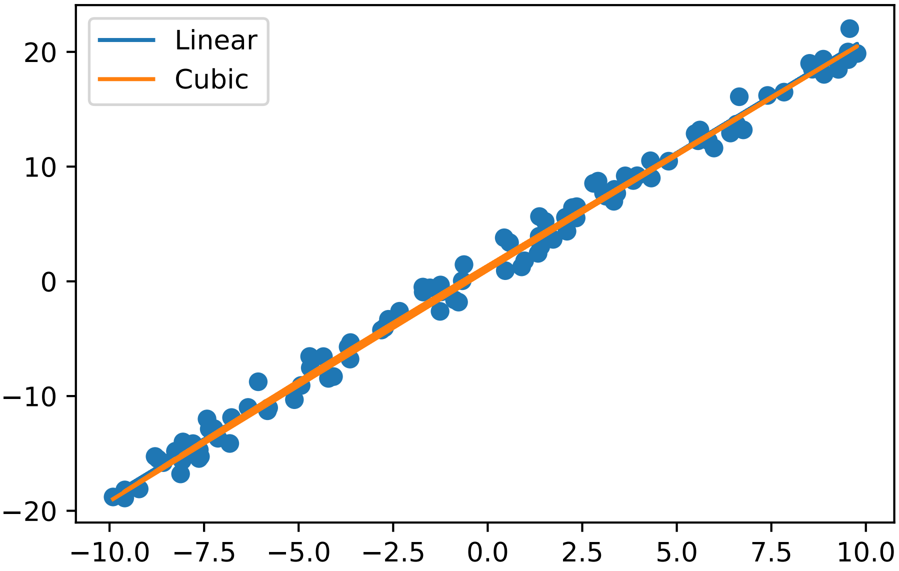
```
Linear Regression: Coefficients and Intercept
[1.99684675] 1.1906185881482492
RSS: 99.24386487246484
   coefficient        se  tstatistic        p-value  r-squared  \
0     1.190619  0.101080   11.779007   1.772355e-20   0.992569
```

```
1     1.996847  0.017453  114.415008  3.862199e-106   0.992569

   pearson_coefficient        rss  sd_residuals
0             0.996278  99.243865      1.006326
1             0.996278  99.243865      1.006326

Cubic Regression: Coefficients and Intercept
[ 1.97357410e+00 -4.59965439e-03  3.26908711e-04] 1.3457300939359265
RSS: 97.11464556716115
   coefficient        se  tstatistic       p-value  r-squared  \
0     1.345730  0.149523    9.000158  2.090625e-14   0.992729
1     1.973574  0.044858   43.996479  1.958057e-65   0.992729
2    -0.004600  0.003312   -1.388809  1.681045e-01   0.992729
3     0.000327  0.000670    0.488215  6.265106e-01   0.992729

   pearson_coefficient        rss  sd_residuals
0             0.996358  97.114646      1.005789
1             0.996358  97.114646      1.005789
2             0.996358  97.114646      1.005789
3             0.996358  97.114646      1.005789
```



Training RSS (Residual Sum of Squares) for Linear and Cubic Regression:

True Model: Linear

y = 2x + 1 +

where ~ N(0, 1)

Dataset:

Generate 100 samples from the true model with x uniformly distributed between -10 and 10.

Linear Regression:

y = 0 + 1x +

Cubic Regression:

y = 0 + 1x + 2x^2 + 3x^3 +

Comparison:

1. Linear Regression:
   - Simple and interpretable model
   - Low bias, high variance
   - RSS: 99.24
2. Cubic Regression:
   - More complex model with non-linear terms
   - Higher bias, lower variance
   - RSS: 97.12

Observations:

1. Cubic regression has a slightly lower RSS, indicating better fit.
2. Linear regression coefficients are closer to true values (0 = 1, 1 = 2).
3. Cubic regression coefficients have higher standard errors.

Overfitting Risk:

Cubic regression may be overfitting, as:

1. Additional terms (x^2, x^3) don't significantly improve fit.
2. Coefficients have high standard errors.

Conclusion:

For this linear true model:

1. Linear regression provides a simple, accurate, and interpretable model.
2. Cubic regression may overfit, despite slightly better fit.

- We can see that the cubic regression is a better fit to the data.
- However, if we look at the p-values for X^2 and X^3, we discern that their coefficents are not significant.
- Thus, the cubic regression overfits to the training data and the *Adjusted* R^2 improves by only 0.001.

(b) Answer (a) using test rather than training RSS.

When we are using test RSS as against the training RSS, the overfitted model i.e., the cubic regression will display more variance for data it has not been exposed to or trained on especially since the true relationship i.e., the population relationship is linear.

```python
# Generate data
x_test = np.random.uniform(-10, 10, 50)
y_test = 2 * x_test + 1 + np.random.normal(0, 1, 50)

# Linear Regression
y_pred_train_lr = lr.predict(x.reshape(-1, 1))
y_pred_test_lr = lr.predict(x_test.reshape(-1, 1))

# Cubic Regressionx_poly_test = poly.transform(x_test.reshape(-1, 1))
x_poly_test = poly.transform(x_test.reshape(-1, 1))
y_pred_train_poly = lr_poly.predict(x_poly)
y_pred_test_poly = lr_poly.predict(x_poly_test)

# Print coefficients and RSS
print("Linear Regression:")
print("Train RSS:", np.sum((y - y_pred_train_lr) ** 2))
print("Test RSS:", np.sum((y_test - y_pred_test_lr) ** 2))

print("\nCubic Regression:")
print("Train RSS:", np.sum((y - y_pred_train_poly) ** 2))
print("Test RSS:", np.sum((y_test - y_pred_test_poly) ** 2))
```

```
Linear Regression:
Train RSS: 99.24386487246484
Test RSS: 46.357511874726775

Cubic Regression:
Train RSS: 97.11464556716115
Test RSS: 46.37737762305997
```

Test RSS (Residual Sum of Squares) evaluation:

Test Dataset:

Generate 50 new samples from the true model with x uniformly distributed between -10 and 10.

Linear Regression:

Test RSS: 46.36

Cubic Regression:

Test RSS: 46.38

*Comparison:

Model	Train RSS	Test RSS
Linear	99.24	46.36
Cubic	97.12	46.38

*Observations:

1. Linear regression generalizes better (lower Test RSS).
2. Cubic regression overfits (higher Test RSS than Train RSS).
3. Linear regression has consistent performance (Train and Test RSS).

*Conclusion:

For this linear true model:

1. Linear regression provides a simple, accurate, and generalizable model.
2. Cubic regression overfits and does not perform as well on unseen data.

- We can conclude that the Test RSS for cubic regression overfits the training data and does not perform as well on unseen data from the population when the true model is linear.
- The Test RSS for the cubic regression exceeds that of the linear regression.

However, since we cannot see this clearly for a single regression model, let's run a simulation of 100 iterations of the regressions and check our results.

```python
def simple_linear_equation():
  def f(x, n_samples):
    y = 2*x + 1 + np.random.normal(0, 1, n_samples)
    return y

  return f

def simulate_regression(y_equation, n_runs = 100,n_samples = 100,test_size = 0.2,x_range = (-10, 10)):
  """
  Simulate regression analysis for a given equation.

  Parameters:
  y_equation (function): Equation for y in terms of x.
  n_runs (int): Number of simulation runs.
  n_samples (int): Number of samples per run.
  test_size (float): Proportion of samples for testing.
  x_range (tuple): Range of x values.

  Returns:
  simulation_results (pandas dataframe): Dataframe containing simulation results.
  """

  # Initialize arrays to store results
  train_rss_linear = np.zeros(n_runs)
  test_rss_linear = np.zeros(n_runs)
  train_rss_cubic = np.zeros(n_runs)
  test_rss_cubic = np.zeros(n_runs)
```

```
p_value_linear = np.zeros(n_runs)
p_value_const = np.zeros(n_runs)
p_value_cubic_const = np.zeros(n_runs)
p_value_linear_cubic = np.zeros(n_runs)
p_value_quadratic = np.zeros(n_runs)
p_value_cubic = np.zeros(n_runs)
r2_adj_linear = np.zeros(n_runs)
r2_adj_cubic = np.zeros(n_runs)

for i in range(n_runs):
  x = np.random.uniform(x_range[0], x_range[1], n_samples)
  y = y_equation(x, n_samples)

  # Split data into training and testing sets
  x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=test_size)

  # Linear Regression
  x_train_sm = sm.add_constant(x_train.reshape(-1, 1))
  model_linear = sm.OLS(y_train, x_train_sm).fit()
  y_pred_train_lr = model_linear.predict(x_train_sm)
  y_pred_test_lr = model_linear.predict(sm.add_constant(x_test.reshape(-1, 1)))

  # Cubic Regression
  poly = PolynomialFeatures(degree=3,include_bias=False)
  x_poly_train = poly.fit_transform(x_train.reshape(-1, 1))
  x_poly_test = poly.transform(x_test.reshape(-1, 1))
  x_poly_train_sm = sm.add_constant(x_poly_train)
  model_cubic = sm.OLS(y_train, x_poly_train_sm).fit()
  y_pred_train_poly = model_cubic.predict(x_poly_train_sm)
  y_pred_test_poly = model_cubic.predict(sm.add_constant(x_poly_test))

  # Store results
  train_rss_linear[i] = np.sum((y_train - y_pred_train_lr) ** 2)
  test_rss_linear[i] = np.sum((y_test - y_pred_test_lr) ** 2)
  train_rss_cubic[i] = np.sum((y_train - y_pred_train_poly) ** 2)
  test_rss_cubic[i] = np.sum((y_test - y_pred_test_poly) ** 2)
  p_value_linear[i] = model_linear.pvalues[1]
  p_value_const[i] = model_linear.pvalues[0]
  p_value_cubic_const = model_cubic.pvalues[0]
  p_value_linear_cubic[i] = model_cubic.pvalues[1]
  p_value_quadratic[i] = model_cubic.pvalues[2]
  p_value_cubic[i] = model_cubic.pvalues[3]
  r2_adj_linear[i] = model_linear.rsquared_adj
  r2_adj_cubic[i] = model_cubic.rsquared_adj
  # results
  dict = {
  "Mean Train RSS (Linear)": np.mean(train_rss_linear),
  "Mean Test RSS (Linear)": np.mean(test_rss_linear),
  "Mean Train RSS (Cubic)": np.mean(train_rss_cubic),
  "Mean Test RSS (Cubic)": np.mean(test_rss_cubic),
  "Mean p-value (Constant Term - Linear)": np.mean(p_value_const),
  "Mean p-value (Linear Term)": np.mean(p_value_linear),
  "Mean p-value (Constant Term - Cubic)": np.mean(p_value_cubic_const),
  "Mean p-value (Linear Term - Cubic)": np.mean(p_value_linear_cubic),
  "Mean p-value (Quadratic Term)": np.mean(p_value_quadratic),
  "Mean p-value (Cubic Term)": np.mean(p_value_cubic),
  "Mean R^2 Adjusted (Linear)": np.mean(r2_adj_linear),
  "Mean R^2 Adjusted (Cubic)": np.mean(r2_adj_cubic)
  }
  return pd.DataFrame(data=dict, index=[0])


n_samples = 1000
simulate_regression(simple_linear_equation())
```

	Mean Train RSS (Linear)	Mean Test RSS (Linear)	Mean Train RSS (Cubic)	Mean Test RSS (Cubic)	Mean p
0	78.26558	21.257098	76.302815	21.825997	1.69847

This still isn't conclusive enough. So we run multiple simulations of different linear models.

```
def linear_equation():
  def f(x, n_samples):
    # Generate linear data with different coefficients in each iteration
    coeff_linear = np.random.uniform(1, 5)
    coeff_const = np.random.uniform(-5, 5)
    y = coeff_linear * x + coeff_const + np.random.normal(0, 1, n_samples)
    return y

  return f

simulate_regression(linear_equation())
```

	Mean Train RSS (Linear)	Mean Test RSS (Linear)	Mean Train RSS (Cubic)	Mean Test RSS (Cubic)	Mean p
0	78.099076	19.010225	75.907893	19.702869	0.02949

(c) Suppose that the true relationship between X and Y is not linear, but we don't know how far it is from linear. Consider the training RSS for cubic regression. Would we expect one to be lower than the linear regression, and also the training RSS for the cubic regression. Would we expect one to be lower than the other, would we expect them to be the same, or is there not enough information to tell? Justify your answer.

(d) Answer (c) using test rather than training RSS.

For Slightly non-linear equation

```
def slightly_non_linear_equation():
  def f(x, n_samples):
    # Generate slightly non-linear data
    coeff_linear = np.random.uniform(1, 5)
    coeff_const = np.random.uniform(-5, 5)
    coeff_nonlinear = np.random.uniform(0.1, 0.5)
    y = coeff_linear * x + coeff_const + coeff_nonlinear * x**2 +
 ↪  np.random.normal(0, 1, n_samples)
    return y

  return f

simulate_regression(slightly_non_linear_equation())
```

	Mean Train RSS (Linear)	Mean Test RSS (Linear)	Mean Train RSS (Cubic)	Mean Test RSS (Cubic)	Mean p
0	6780.349659	1873.954548	76.792125	21.66511	0.00323

Mostly non-linear equation

```
def mostly_non_linear_equation():
  def f(x, n_samples):
    # Generate very non-linear data
    coeff_linear = np.random.uniform(1, 5)
    coeff_const = np.random.uniform(-5, 5)
    coeff_nonlinear1 = np.random.uniform(0.5, 2)
    coeff_nonlinear2 = np.random.uniform(0.2, 1)
    y = coeff_linear * x + coeff_const + coeff_nonlinear1 * np.sin(x) +
 ↪  coeff_nonlinear2 * x**3 + np.random.normal(0, 1, n_samples)
    return y

  return f

simulate_regression(mostly_non_linear_equation())
```

	Mean Train RSS (Linear)	Mean Test RSS (Linear)	Mean Train RSS (Cubic)	Mean Test RSS (Cubic)	Mean p
0	723145.034329	196768.009208	137.519661	37.215332	0.49817

- From the above, we can conclude that whether the original true model is slightly non-linear or heavily non-linear, the cubic regression's RSS for both training and test datasets are lower than the linear model's.
- Whether the coeffcients of the cubic model's fit are significant or not depends on the form of the original equation and whether that term exists in the true model or whether the cubic regression's terms are able to approximate that fit despite not matching the original equation exactly. Here, the cubic term with power of three is able to approximate the osciallation of the sine transformation of the input variable.

5. Consider the fitted values that result from performing linear regression without an intercept. In this setting, the i_{th} fitted value takes the form

$\hat{y}_i = x_i * \hat{\beta}$

where

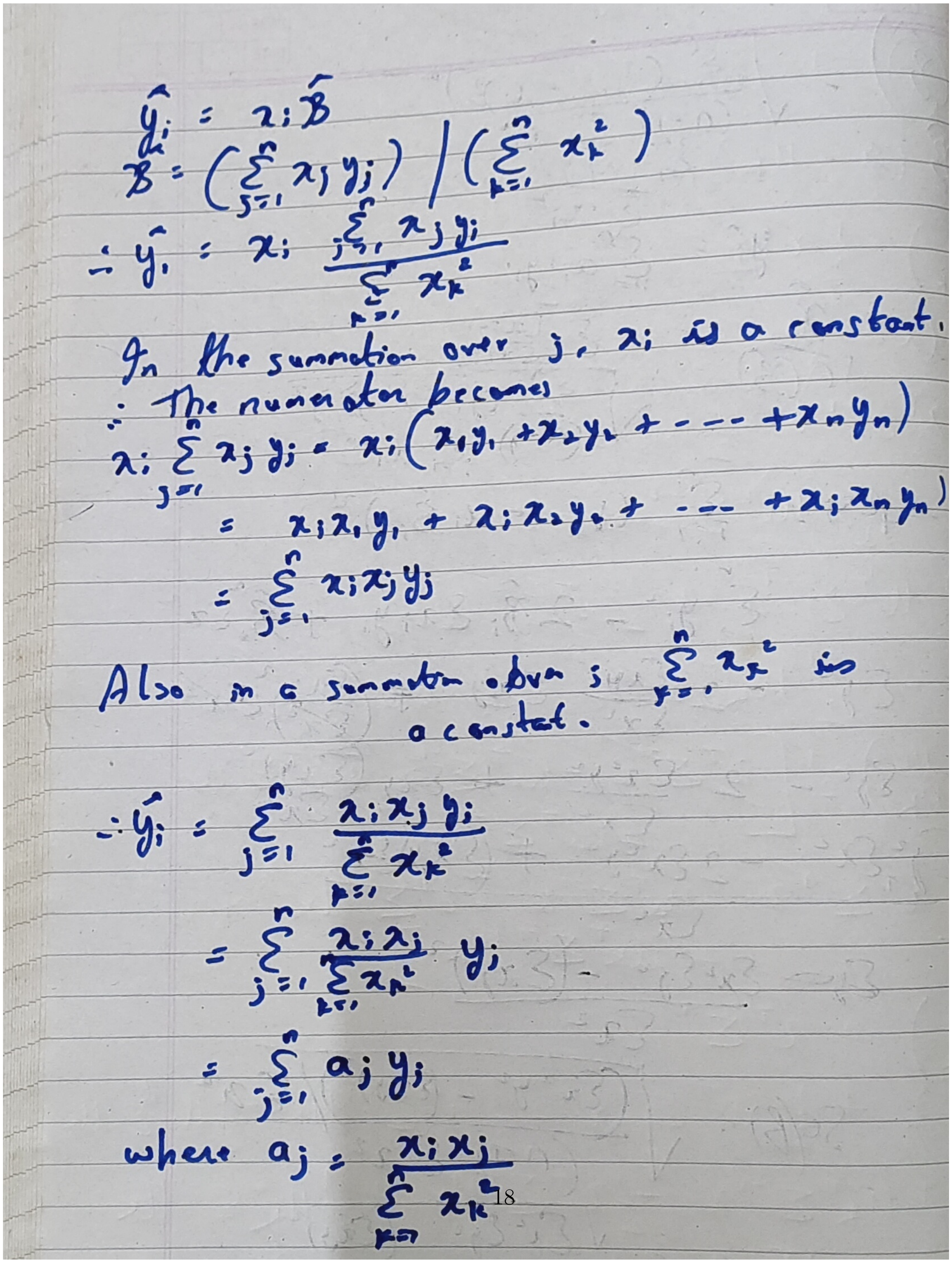
$\hat{\beta} = (\sum_{i=1}^{n} x_i y_i)/(\sum_{i'=1}^{n} x_{i'}^2)$

Show that we can write

$\hat{y}_i = \sum_{i'=1}^{n} a_{i'} y_{i'}$

What is $a_{i'}$?

Note: We interpret this result by saying that the fitted values from linear regression are linear combinations of the response values.



$$\hat{y}_i = x_i \hat{\beta}$$

$$\hat{\beta} = \left(\sum_{j=1}^{n} x_j y_j\right) \Big/ \left(\sum_{k=1}^{n} x_k^2\right)$$

$$\therefore \hat{y}_i = x_i \frac{\sum_{j=1}^{n} x_j y_j}{\sum_{k=1}^{n} x_k^2}$$

In the summation over j, x_i is a constant.

$\therefore$ The numerator becomes

$$x_i \sum_{j=1}^{n} x_j y_j = x_i(x_1 y_1 + x_2 y_2 + --- + x_n y_n)$$

$$= x_i x_1 y_1 + x_i x_2 y_2 + --- + x_i x_n y_n)$$

$$= \sum_{j=1}^{n} x_i x_j y_j$$

Also in a summation over j, $\sum_{k=1}^{n} x_k^2$ is a constant.

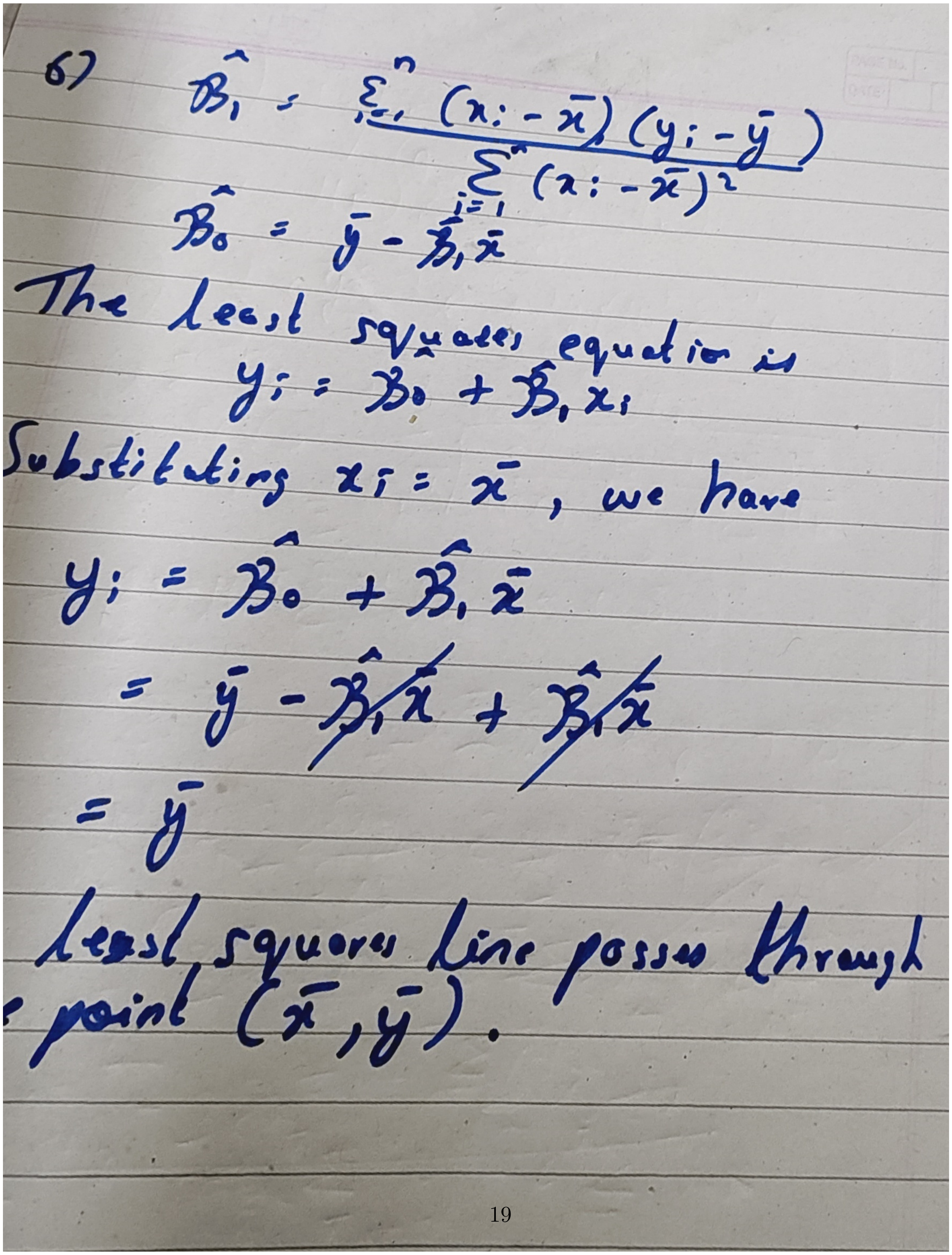
$$\therefore \hat{y}_i = \sum_{j=1}^{n} \frac{x_i x_j y_i}{\sum_{k=1}^{n} x_k^2}$$

$$= \sum_{j=1}^{n} \frac{x_i x_j}{\sum_{k=1}^{n} x_k^2} y_j$$

$$= \sum_{j=1}^{n} a_j y_j$$

where $a_j = \dfrac{x_i x_j}{\sum_{k=1}^{n} x_k^2}$

6. Using (3.4), argue that in the case of simple linear regression, the least squares line always passes through the point $(\bar{x}, \bar{y})$.



6) $$\hat{\beta}_1 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

The least squares equation is

$$y_i = \beta_0 + \hat{\beta}_1 x_i$$

Substituting $x_i = \bar{x}$, we have

$$y_i = \hat{\beta}_0 + \hat{\beta}_1 \bar{x}$$

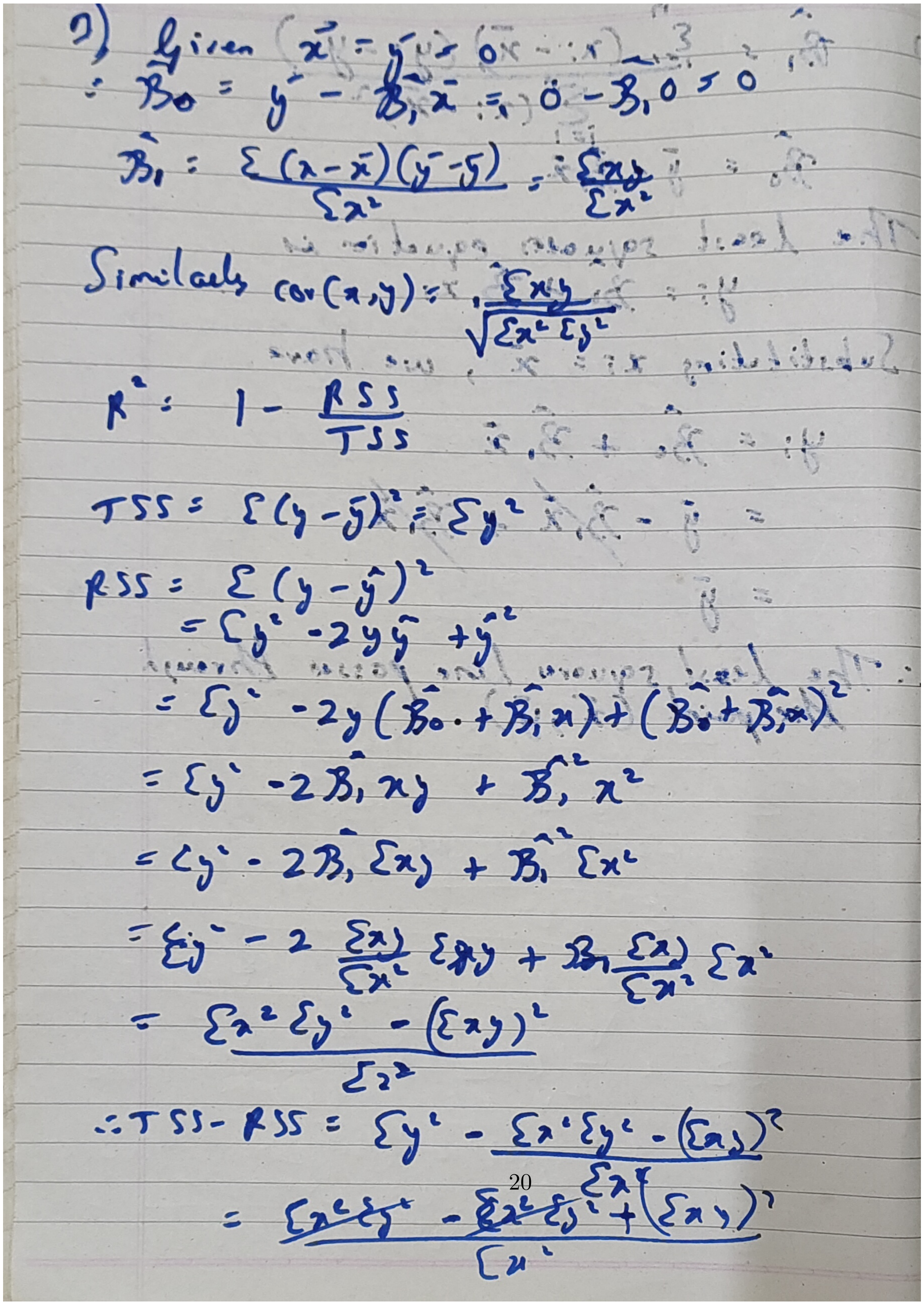
$$= \bar{y} - \cancel{\hat{\beta}_1 \bar{x}} + \cancel{\hat{\beta}_1 \bar{x}}$$

$$= \bar{y}$$

least squares line passes through

e point $(\bar{x}, \bar{y})$.

7. It is claimed in the text that in the case of simple linear regression of Y onto X, the R^2 statistic (3.17) is equal to the square of the correlation between X and Y (3.18). Prove that this is the case. For simplicity, you may assume that $\bar{x} = \bar{y} = 0$.



7) Given $\bar{x} = \bar{y} = 0$

$\therefore \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x} = 0 - \hat{\beta}_1 0 = 0$

$\hat{\beta}_1 = \frac{\sum (x-\bar{x})(y-\bar{y})}{\sum x^2} = \frac{\sum xy}{\sum x^2}$

Similarly $cor(x,y) = \frac{\sum xy}{\sqrt{\sum x^2 \sum y^2}}$

$R^2 = 1 - \frac{RSS}{TSS}$

$TSS = \sum (y-\bar{y})^2 = \sum y^2$

$RSS = \sum (y-\hat{y})^2$

$= \sum y^2 - 2y\hat{y} + \hat{y}^2$

$= \sum y^2 - 2y(\hat{\beta}_0 + \hat{\beta}_1 x) + (\hat{\beta}_0 + \hat{\beta}_1 x)^2$

$= \sum y^2 - 2\hat{\beta}_1 xy + \hat{\beta}_1^2 x^2$

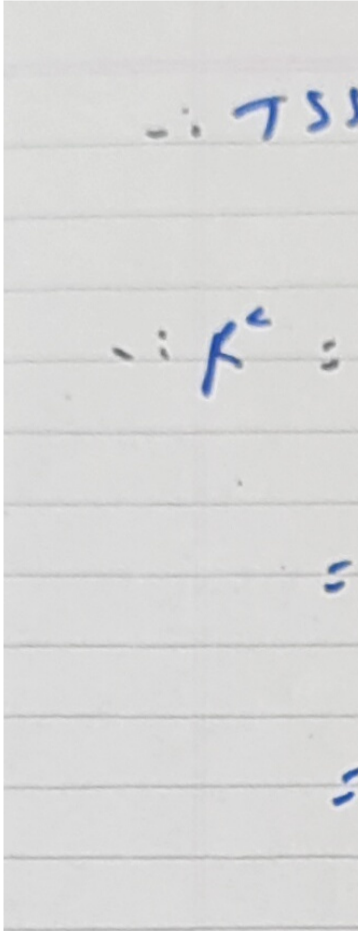
$= \sum y^2 - 2\hat{\beta}_1 \sum xy + \hat{\beta}_1^2 \sum x^2$

$= \sum y^2 - 2 \frac{\sum xy}{\sum x^2} \sum xy + \hat{\beta}_1 \frac{\sum xy}{\sum x^2} \sum x^2$

$= \frac{\sum x^2 \sum y^2 - (\sum xy)^2}{\sum x^2}$

$\therefore TSS - RSS = \sum y^2 - \frac{\sum x^2 \sum y^2 - (\sum xy)^2}{\sum x^2}$

$= \frac{\cancel{\sum x^2 \sum y^2} - \cancel{\sum x^2 \sum y^2} + (\sum xy)^2}{\sum x^2}$



$\therefore TS$

$\therefore R^2 =$

$=$

$=$

Standardized regression coefficients can be calculated from the original regression coefficients and the standard deviations of the original coefficients and the response variable.

$$b_k = b_{k'} * \frac{s_{x_k}}{s_y}$$

Step	Rationale
$Y = a + b_1 X_1 + b_2 X_2 + e \implies Y - \bar{y} = a + b_1 X_1 + b_2 X_2 + e - \bar{y}$	Subtract $\bar{y}$ from both sides
$= \bar{y} - b_1 \bar{X_1} - b_2 \bar{X_2} + b_1 X_1 + b_2 X_2 + e - \bar{y}$	Substitute for a
$= b_1 (X1 - \bar{X_1}) + b_2 (X2 - \bar{X}2) + e$	Rearrange terms
$= b_1 * s_1 * \frac{(X_1 - \bar{X_1})}{s_1} + b_2 * s_2 * \frac{(X_2 - \bar{X_2})}{s_2} + e$	Multiply and divide by s.d's
$= b_1 * s_1 * X_1^{'} + b_2 * s_2 * X_2^{'} + e$	Substitute standardized X's
$\implies \frac{(Y - \bar{y})}{s_y} = Y^{'} = b_1 * \frac{s_1}{s_y} * X_1^{'} + b_2 * \frac{s_2}{s_y} * X_2^{'} + \frac{e}{s_y}$	Divide both sides by s_y
$= b_1^{'} X_1^{'} + b_2^{'} X_2^{'} + e^{'}$	Substitute standardized coefficients
$\implies b_k^{'} = b_k * \frac{s_k}{s_y}$	Q.E.D (that which was to be shown)

References: 1. https://www3.nd.edu/~rwilliam/stats1/x92.pdf

```
allDone();
```

```
<IPython.lib.display.Audio object>
```