Data Mining (KEN4113)

Lab 1: Regression

Konstantin Sandfort i6255681

In [107…

```
# Imports
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn import metrics
from sklearn.preprocessing import PolynomialFeatures
```

1. Linear Regression Model Analysis

1.1 Gender Bias Analysis

In [108…

```
b_0 = 50
b_1 = 20
b_2 = 0.07
b_3 = 10
b_4 = 0.01
b_5 = -3

# Vector of coefficients
b = [b_0, b_1, b_2, b_3, b_4, b_5]

def calc_salary(person, coeff):
    """
    Calculates the estimated salary of a person according to the regression
    :param person: Feature Vector X (length = 3)
    :param coeff: Coefficient Vector beta (length = 5)
    :return: estimated yearly salary in 1000 euros
    """
    salary = coeff[0] + (person[0] * coeff[1]) + (person[1] * coeff[2]) + (p
    # Add interaction terms
    salary += person[0] * person[1] * coeff[4]
    salary += person[0] * person[2] * coeff[5]
    return salary

# Generate 1000 male and female individuals with random GPA and IQ
# We don't know anything about the distribution, so we assume a normal distr
n = 10000
# Generate random numbers from a normal distribution
male_gpa = np.random.normal(2, 1, n)
male_iq = np.random.normal(100, 15, n)

female_gpa = np.random.normal(2, 1, n)
```

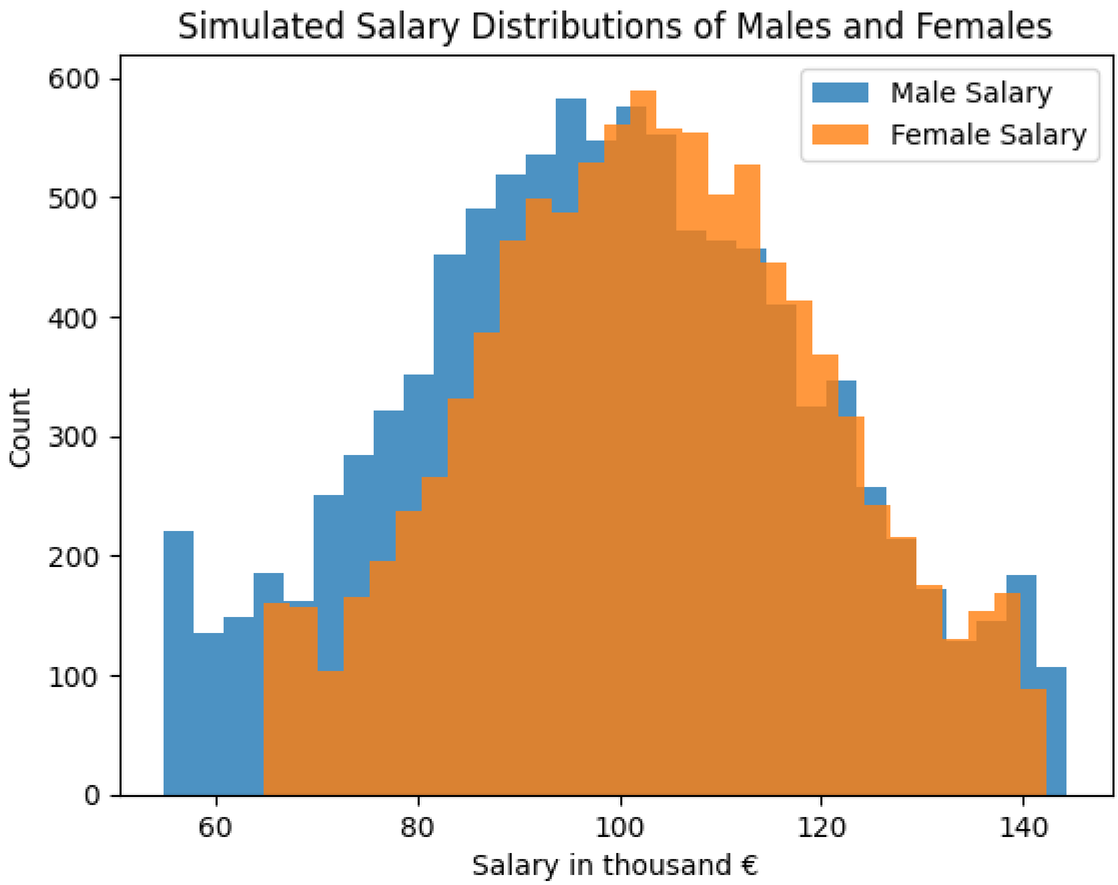
```python
female_iq = np.random.normal(100, 15, n)

def squish_to_bounds(x, lower, upper):
    """
    This function squishes a value to a certain range defined by a lower and
    :param x: target value
    :param lower: lower bound
    :param upper: upper bound
    :return: squished value
    """
    if x < lower:
        return lower
    elif x > upper:
        return upper
    else:
        return x

# Vectorise function
squish_vec = np.vectorize(squish_to_bounds)

# Squish values to range
male_gpa_adjusted = squish_vec(male_gpa, 0, 4)
male_iq_adjusted = squish_vec(male_iq, 70, 130)

female_gpa_adjusted = squish_vec(female_gpa, 0, 4)
female_iq_adjusted = squish_vec(female_iq, 70, 130)

males = np.transpose(np.vstack([male_gpa_adjusted, male_iq_adjusted, np.zero
females = np.transpose(np.vstack([female_gpa_adjusted, female_iq_adjusted, n

# The following two vectors contain n salaries of each individual.
salary_male = np.apply_along_axis(calc_salary, 1, males, b)
salary_female = np.apply_along_axis(calc_salary, 1, females, b)
```

In [109…

```python
# Plot histograms and show salary distribution properties
plt.hist(salary_male, bins=30, alpha=0.8, label='Male Salary')
plt.hist(salary_female, bins=30, alpha= 0.8, label='Female Salary')
plt.title('Simulated Salary Distributions of Males and Females')
plt.xlabel('Salary in thousand €')
plt.ylabel('Count')
plt.legend()
```

Out[109…

```
<matplotlib.legend.Legend at 0x7fe74a11a260>
```



In [110…

```
# Compute mean and std deviation of both distributions
male_mean = np.mean(salary_male)
male_std = np.std(salary_male)

female_mean = np.mean(salary_female)
female_std = np.std(salary_female)

print(f'Male Distribution: Mean = {male_mean}, STD = {male_std}')
print(f'Female Distribution: Mean = {female_mean}, STD = {female_std}')
```

```
Male Distribution: Mean = 98.74691106711434, STD = 20.28062172629942
Female Distribution: Mean = 103.11876023628226, STD = 17.203907426084207
```

1.2 Model Tree

(Included in the analytic report)

2. Data Generation and Model Fitting

2.1 First Data Generation

In [111…

```
# Set random seed
np.random.seed(42)

# Generate random vectors
```

```python
x = np.random.normal(0, 1, 100)
eps = np.random.normal(0, np.sqrt(0.25), 100)
y = -0.5 + (0.75 * x) + eps
print(f'Vector y length: {len(y)}')
```

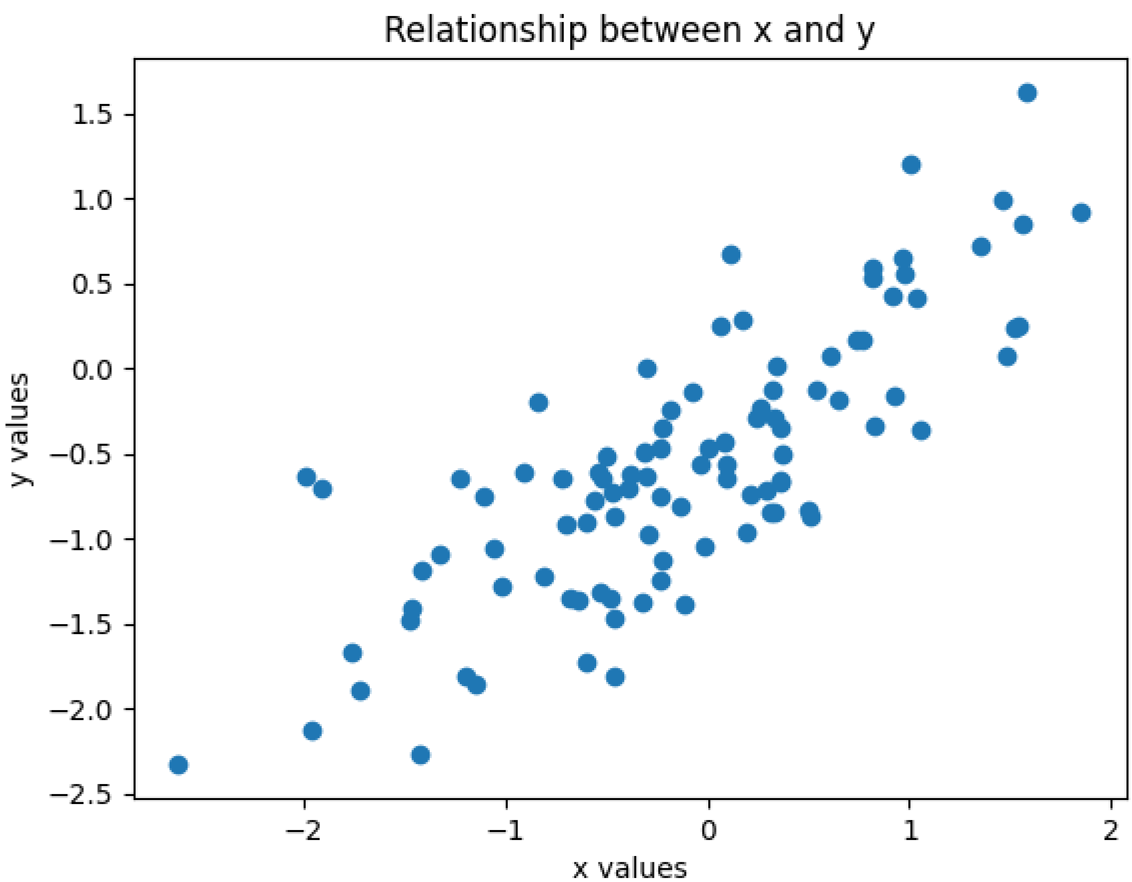
```
Vector y length: 100
```

2.2 First Data Visualisation

In [112…

```python
# Generate scatter plot
plt.scatter(x, y)
plt.title('Relationship between x and y')
plt.xlabel('x values')
plt.ylabel('y values')
```

Out[112… Text(0, 0.5, 'y values')



2.3 Fitting First Linear Regression

In [113…

```python
# (a) Create linear regression
model = linear_model.LinearRegression()
model.fit(x[:, None], y)
print('Linear Regression model coefficients')
print(f'b_0: {model.intercept_}, b_1: {model.coef_[0]}')
```

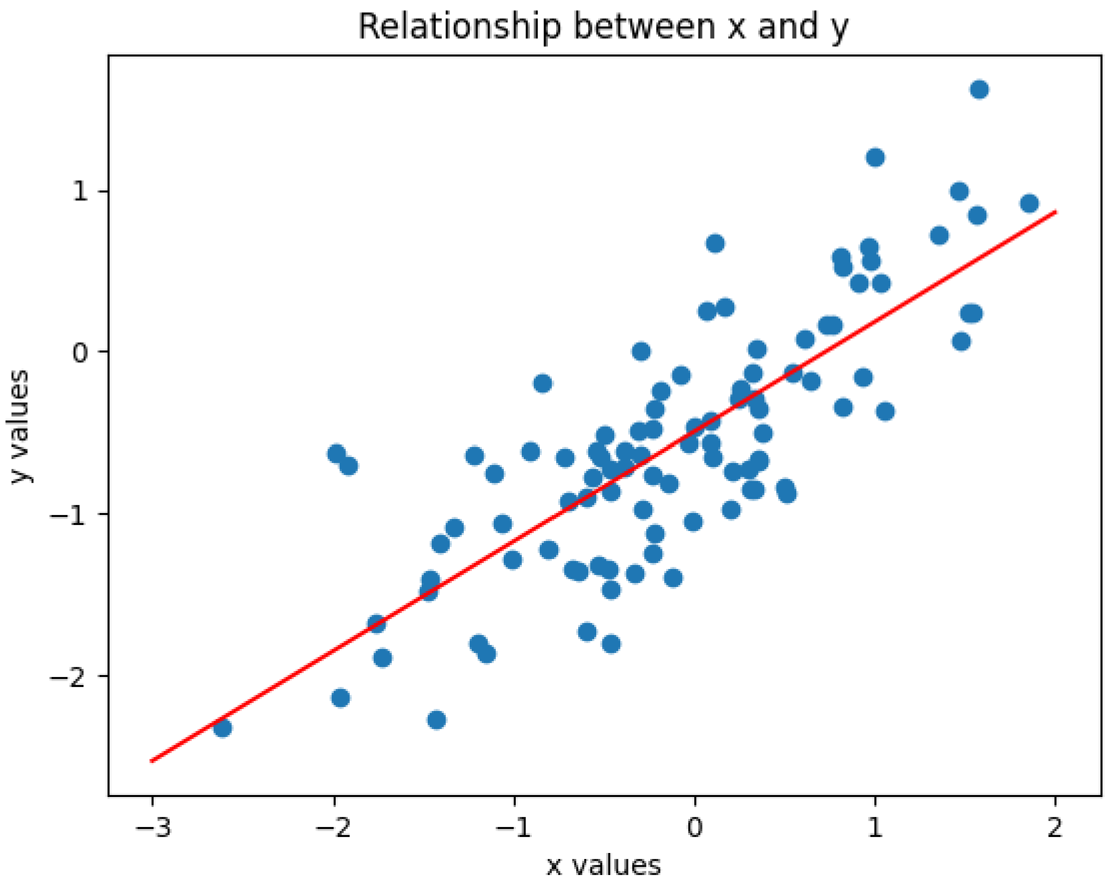
```
Linear Regression model coefficients
b_0: -0.4962860850680164, b_1: 0.6783714198642783
```

In [114…

```python
# (b) Plot the same scatter plot with regression line:
plt.scatter(x, y)
plt.title('Relationship between x and y')
plt.xlabel('x values')
plt.ylabel('y values')
x_line = np.linspace(-3, 2, 100)
y_line = (model.intercept_ + model.coef_[0] * x_line)
plt.plot(x_line, y_line, color='red')
```

Out[114…

```
[<matplotlib.lines.Line2D at 0x7fe7498c3a90>]
```



In [115…

```python
# (c) Calculate R^2 statistics
y_pred = model.intercept_ + (x * model.coef_[0])
r2 = metrics.r2_score(y, y_pred)
print(f'r2 score: {r2}')
```

```
r2 score: 0.6297598193059208
```

2.4 Fitting Second Linear Regression

In [116…

```python
# (a), (b) Create polynomial regression model model
poly_model = PolynomialFeatures(degree=2, include_bias=False)
features = poly_model.fit_transform(x[:, None])
poly_linear_model = linear_model.LinearRegression()
poly_linear_model.fit(features, y)
print('Polynomial Regression model coefficients')
print(f'b_0: {poly_linear_model.intercept_}, b_1: {poly_linear_model.coef_[0
```

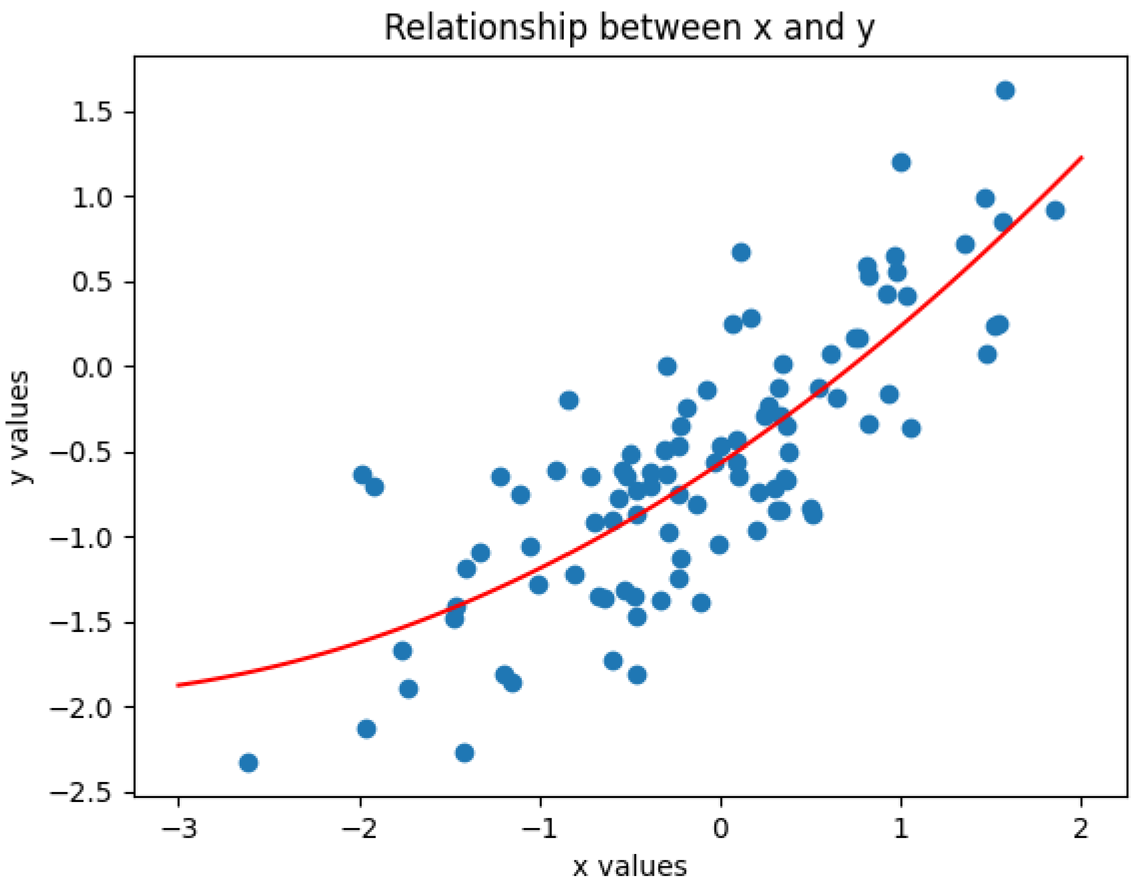
```
Polynomial Regression model coefficients
b_0: -0.5690705740231349, b_1: 0.7121283510062472, b_2: 0.09221497437831454
```

In [117…

```
# (c) Generate new scatter plot
plt.scatter(x, y)
plt.title('Relationship between x and y')
plt.xlabel('x values')
plt.ylabel('y values')
x_line = np.linspace(-3, 2, 100)
y_line = (poly_linear_model.intercept_ + (poly_linear_model.coef_[0] * x_lir
plt.plot(x_line, y_line, color='red')
```

Out[117…

```
[<matplotlib.lines.Line2D at 0x7fe74973e650>]
```



In [118…

```
# (d) Calculate R^2 statistics
y_pred = poly_linear_model.predict(features)
r2 = metrics.r2_score(y, y_pred)
print(f'r2 score: {r2}')
```

```
r2 score: 0.6469951045504286
```

2.5 Second Data Generation

In [119…

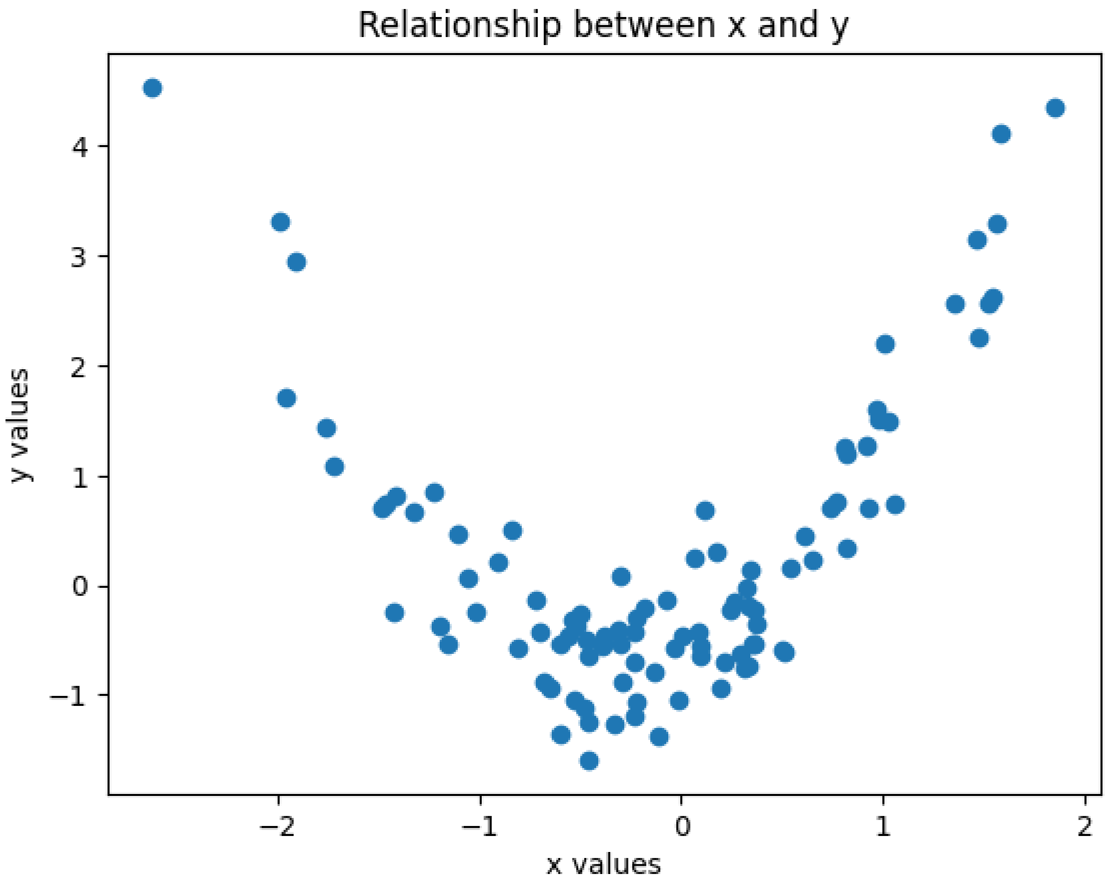
```
# Alter y values to quadratic function
y = -0.5 + (0.75 * x) + x**2 + eps
```

2.6 Second Data Visualisation

In [120…

```
# Generate new scatter plot
plt.scatter(x, y)
plt.title('Relationship between x and y')
plt.xlabel('x values')
plt.ylabel('y values')
```

Out[120… Text(0, 0.5, 'y values')



2.7 Fitting Third Linear Regression

In [121…

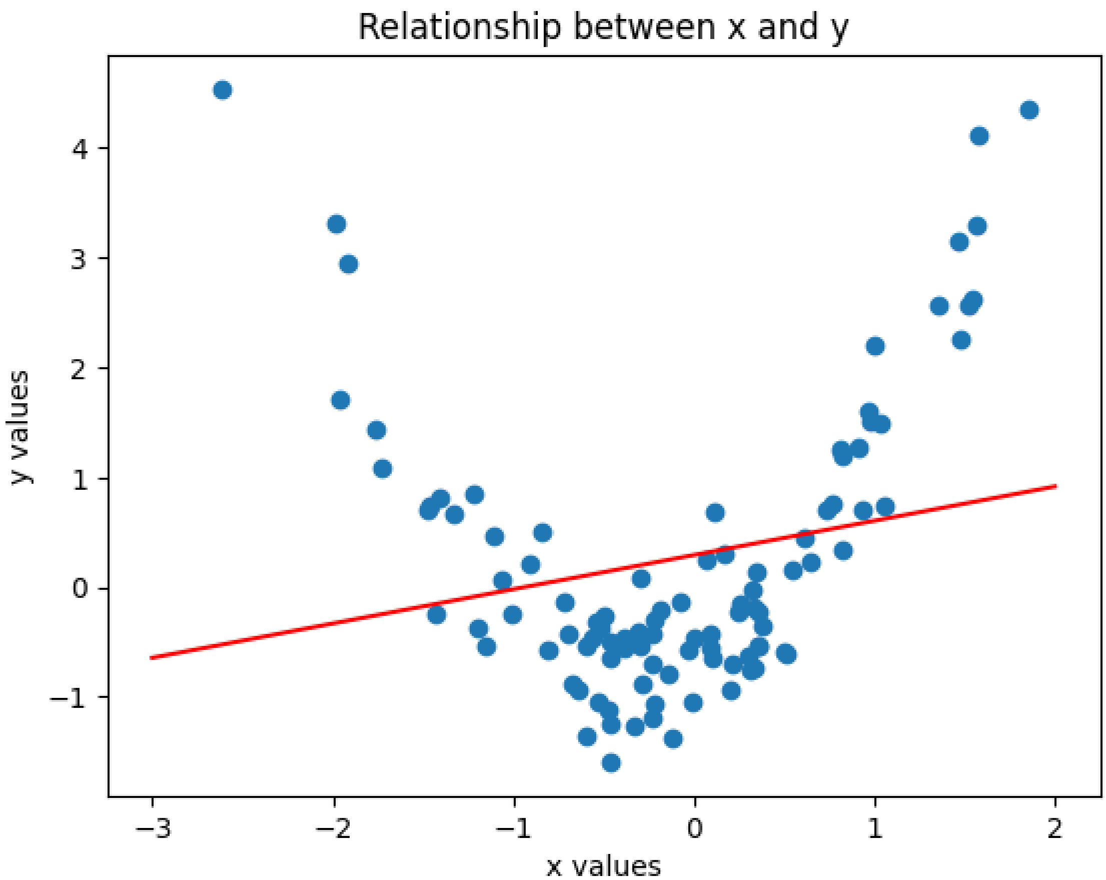
```
# (a) Create new regression model
model = linear_model.LinearRegression()
model.fit(x[:, None], y)
print('Linear Regression model coefficients')
print(f'b_0: {model.intercept_}, b_1: {model.coef_[0]}')
```

```
Linear Regression model coefficients
b_0: 0.2930053445052121, b_1: 0.31230363781968196
```

In [122…

```
# (b) Plot the same scatter plot with least squares line
plt.scatter(x, y)
plt.title('Relationship between x and y')
plt.xlabel('x values')
plt.ylabel('y values')
x_line = np.linspace(-3, 2, 100)
y_line = (model.intercept_ + model.coef_[0] * x_line)
plt.plot(x_line, y_line, color='red')
```

Out[122… [<matplotlib.lines.Line2D at 0x7fe74976e410>]



In [123…
```
# (c) Calculate R^2 statistics
y_pred = model.predict(x[:, None])
r2 = metrics.r2_score(y, y_pred)
print(f'r2 score: {r2}')
```

```
r2 score: 0.045956423052825435
```

2.8 Fitting Fourth Linear Regression

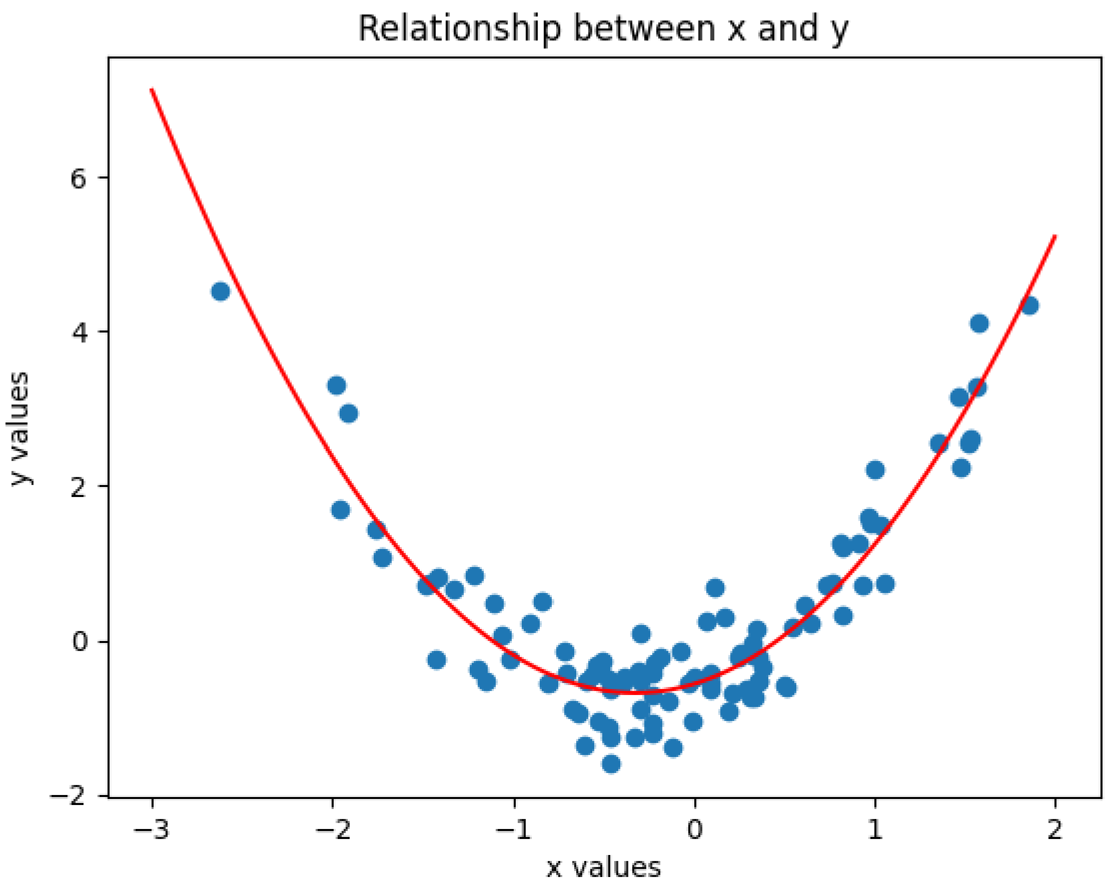
In [124…
```
# (a) Create new regression model
poly_model = PolynomialFeatures(degree=2, include_bias=False)
features = poly_model.fit_transform(x[:, None])
poly_linear_model = linear_model.LinearRegression()
poly_linear_model.fit(features, y)
print('Polynomial Regression model coefficients')
print(f'b_0: {poly_linear_model.intercept_}, b_1: {poly_linear_model.coef_[0
```

```
Polynomial Regression model coefficients
b_0: -0.5690705740231348, b_1: 0.7121283510062472, b_2: 1.0922149743783147
```

In [125…
```
# Generate scatter plot
plt.scatter(x, y)
plt.title('Relationship between x and y')
plt.xlabel('x values')
plt.ylabel('y values')
x_line = np.linspace(-3, 2, 100)
```

```
y_line = (poly_linear_model.intercept_ + (poly_linear_model.coef_[0] * x_lir
plt.plot(x_line, y_line, color='red')
```

Out[125… [<matplotlib.lines.Line2D at 0x7fe74963f3d0>]



In [126…
```
# (c) Calculate R^2 statistics
y_pred = poly_linear_model.predict(features)
r2 = metrics.r2_score(y, y_pred)
print(f'r2 score: {r2}')
```

```
r2 score: 0.8784561474099984
```

3 LASSO Regression Model Analysis

In [127…
```
# Reset y to linear function
y = -0.5 + (0.75 * x) + eps

# Create LASSO model
l1_model = linear_model.Lasso(alpha=0.5)
l1_model.fit(x[:, None], y)
print('LASSO Regression model coefficients')
print(f'b_0: {l1_model.intercept_}, b_1: {l1_model.coef_[0]}')
```

```
LASSO Regression model coefficients
b_0: -0.5598768350004352, b_1: 0.06601819389046765
```

In [128…
```
# Use quadratic function
y = -0.5 + (0.75 * x) + x**2 + eps
```

```
# Create LASSO model
l1_model = linear_model.Lasso(alpha=0.5)
l1_model.fit(x[:, None], y)
print('LASSO Regression model coefficients')
print(f'b_0: {l1_model.intercept_}, b_1: {l1_model.coef_[0]}')
```

```
LASSO Regression model coefficients
b_0: 0.26057369934813174, b_1: 0.0
```

In [129…

```
# Plot scatter plot with regression line:
plt.scatter(x, y)
plt.title('Relationship between x and y')
plt.xlabel('x values')
plt.ylabel('y values')
x_line = np.linspace(-3, 2, 100)
y_line = (model.intercept_ + model.coef_[0] * x_line)
plt.plot(x_line, y_line, color='red')
```

Out[129…

```
[<matplotlib.lines.Line2D at 0x7fe7496d43a0>]
```