Appendix

Data set 1 is used for this project

```
from bokeh.plotting import figure, show
from math import sqrt
import numpy as np
from bokeh.transform import linear_cmap
import pandas as pd
from sqlalchemy import create_engine, Column, Integer, Float
from sqlalchemy.orm import declarative_base, sessionmaker

engine = create_engine('sqlite:///iutask.db')
Session = sessionmaker(bind=engine)
session = Session()

Base = declarative_base()


class TrainTable(Base):
    """
    Represents the TrainTable table in the database.
    """

    __tablename__ = 'TrainTable'

    id = Column(Integer, primary_key=True)
    x = Column(Float)
    y1 = Column(Float)
    y2 = Column(Float)
    y3 = Column(Float)
    y4 = Column(Float)


TrainTable.__table__.drop(engine, checkfirst=True)
Base.metadata.create_all(engine)

df = pd.read_csv('train.csv')
df.to_sql('TrainTable', engine, if_exists='append', index=False)

print("Expected Structure of TrainTable:")
print("ID\tx\ty1\ty2\ty3\ty4")


class IdealTable(Base):
    """SQLAlchemy model for IdealTable."""
    __tablename__ = 'IdealTable'
    id = Column(Integer, primary_key=True)
    x = Column(Float)
    for i in range(1, 51):
        locals()[f'y{i}'] = Column(Float)
```

```
IdealTable.__table__.drop(engine, checkfirst=True)
Base.metadata.create_all(engine)

df = pd.read_csv('ideal.csv')
df.to_sql('IdealTable', engine, if_exists='append', index=False)

print("Expected Structure of IdealTable:")
print("ID\tx\ty1\ty2\ty3\ty4\ty5...\ty50")

query = session.query(TrainTable)
df = pd.DataFrame([(row.id, row.x, row.y1, row.y2, row.y3, row.y4) for row in query],
                  columns=['id', 'x', 'y1', 'y2', 'y3', 'y4'])

p = figure(width=800, height=400, title="TrainTable (y1 to y4)")
mapper = linear_cmap(field_name="y", palette="Viridis256", low=df['y1'].min(),
high=df['y4'].max())

p.line(x='x', y='y1', source=df, color='blue', line_width=2, legend_label="y1")
p.line(x='x', y='y2', source=df, color='green', line_width=2, legend_label="y2")
p.line(x='x', y='y3', source=df, color='orange', line_width=2, legend_label="y3")
p.line(x='x', y='y4', source=df, color='red', line_width=2, legend_label="y4")
p.xaxis.axis_label = "x"
p.yaxis.axis_label = "y"
p.add_layout(p.legend[0], 'right')

show(p)


def least_squares_difference(y1, y2):
    """
    Calculates the least squares difference between two arrays.

    Parameters:
    - y1 (numpy.ndarray): The first array.
    - y2 (numpy.ndarray): The second array.

    Returns:
    - float: The least squares difference between the arrays.
    """
    return np.sum((y1 - y2) ** 2)

nearest_ideal_function_indices = []
min_ls_differences = []

for column_name in ['y1', 'y2', 'y3', 'y4']:
    train_y_column = np.array([getattr(train_row, column_name) for train_row in
session.query(TrainTable)])

    ideal_y_columns = np.array([[getattr(ideal_row, f'y{i}')
```

```
                                for ideal_row in session.query(IdealTable)] for i in
range(1, 51)])

    min_ls_difference = float('inf')
    nearest_ideal_function_index = None

    for i, ideal_y_column in enumerate(ideal_y_columns):
        ls_difference = least_squares_difference(train_y_column, ideal_y_column)
        if ls_difference < min_ls_difference:
            min_ls_difference = ls_difference
            nearest_ideal_function_index = i

    nearest_ideal_function_indices.append(nearest_ideal_function_index)
    min_ls_differences.append(min_ls_difference)

selected_indices = nearest_ideal_function_indices

x_values = np.array([getattr(train_row, 'x') for train_row in session.query(TrainTable)])

p = figure(width=800, height=400, title="Selected Ideal Functions")

colors = ['blue', 'green', 'orange', 'red']

for i, idx in enumerate(selected_indices):
    y_values = np.array([getattr(ideal_row, f'y{idx+1}') for ideal_row in
session.query(IdealTable)])
    p.line(x=x_values, y=y_values, color=colors[i], line_width=2, legend_label=f'Ideal
Function {idx+1}')

p.xaxis.axis_label = "x"
p.yaxis.axis_label = "y"
p.add_layout(p.legend[0], 'right')

show(p)

test_df = pd.read_csv('test.csv')
x_test = test_df['x']
y_test = test_df['y']

sum_of_deviations = []

for ideal_idx_global in selected_indices:
    sum_deviation = 0
    for idx, x_value in enumerate(x_test):
        y_test_value = y_test[idx]
        ideal_y_values = np.array([getattr(ideal_row,
                                           f'y{ideal_idx_global + 1}') for ideal_row in
session.query(IdealTable)])
        ideal_idx = np.argmin(np.abs(x_values - x_value))
        deviation = np.abs(ideal_y_values[ideal_idx] - y_test_value)
        sum_deviation += deviation
```

```
    sum_of_deviations.append(sum_deviation)

best_fit_index = np.argmin(sum_of_deviations)

# Determine if the deviation exceeds the threshold
threshold = max(sum_of_deviations) * sqrt(2)

if sum_of_deviations[best_fit_index] > threshold:
    print("Warning: Maximum deviation exceeds threshold. Choosing a different ideal
function.")

print(f"Best fit ideal function index: {selected_indices[best_fit_index]}, "
      f"Sum of deviations: {sum_of_deviations[best_fit_index]}")

p = figure(width=800, height=400, title="Test Function vs. Selected Ideal Function")

p.scatter(x=x_test, y=y_test, color='black', legend_label="Test Function")

ideal_idx_global = selected_indices[best_fit_index]
ideal_y_values = np.array([getattr(ideal_row, f'y{ideal_idx_global+1}') for ideal_row in
session.query(IdealTable)])
p.line(x=x_values, y=ideal_y_values, color='blue', line_width=2, legend_label=f'Selected
Ideal '
                                                                        f'Function
{ideal_idx_global+1}')

p.xaxis.axis_label = "x"
p.yaxis.axis_label = "y"
p.add_layout(p.legend[0], 'right')

show(p)


class TestResult(Base):
    """
    Represents the TestResult table in the database.
    """

    __tablename__ = 'TestResult'

    id = Column(Integer, primary_key=True)
    x_test = Column(Float)
    y_test = Column(Float)
    y_ideal = Column(Float)
    deviation = Column(Float)


TestResult.__table__.drop(engine, checkfirst=True)
Base.metadata.create_all(engine)

y_ideal_values = np.array([getattr(ideal_row, f'y{ideal_idx_global+1}') for ideal_row in
```

```
session.query(IdealTable)])
y_ideal_test = np.array([y_ideal_values[np.argmin(np.abs(x_values - x_value))] for x_value in
x_test])
deviations = np.abs(y_test - y_ideal_test)

for i, x_value in enumerate(x_test):
    new_entry = TestResult(x_test=x_value, y_test=y_test[i], y_ideal=y_ideal_test[i],
deviation=deviations[i])
    session.add(new_entry)

session.commit()
session.close()
```

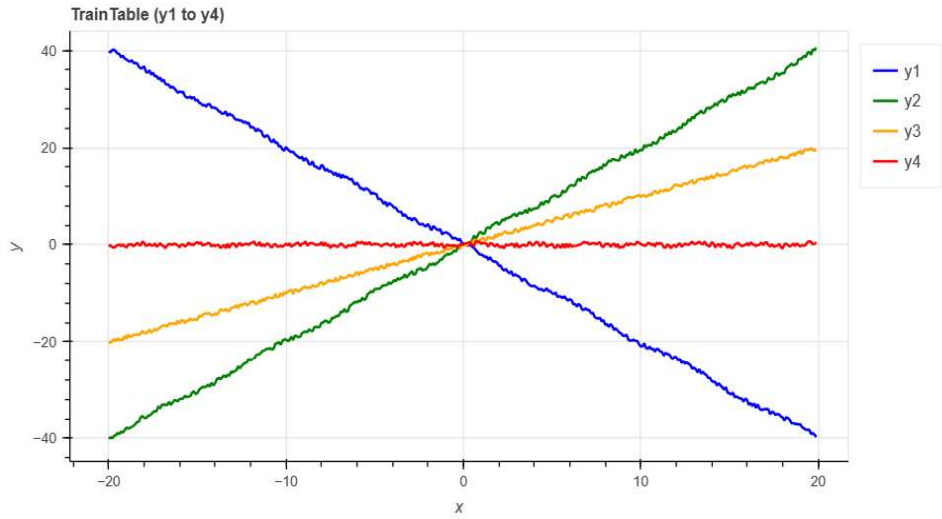

Figure 1 : Representation of train data

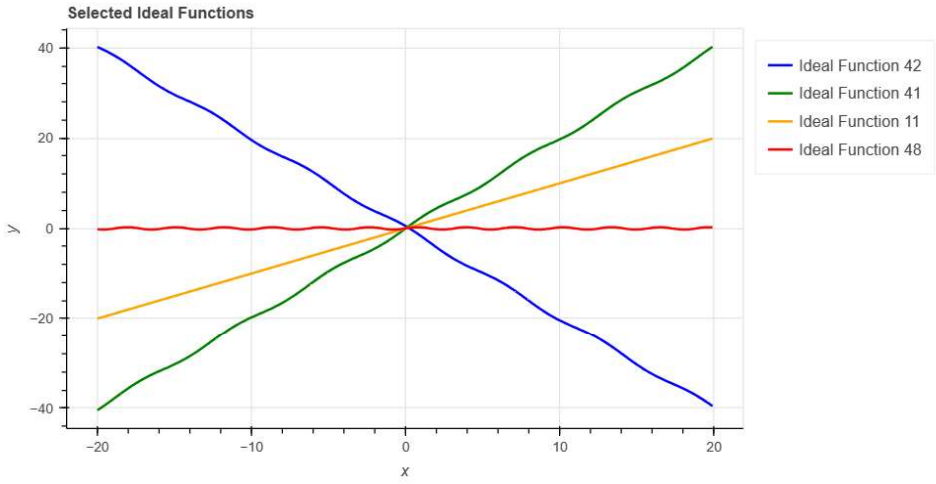

Figure 2 : Representation of selected ideal functions

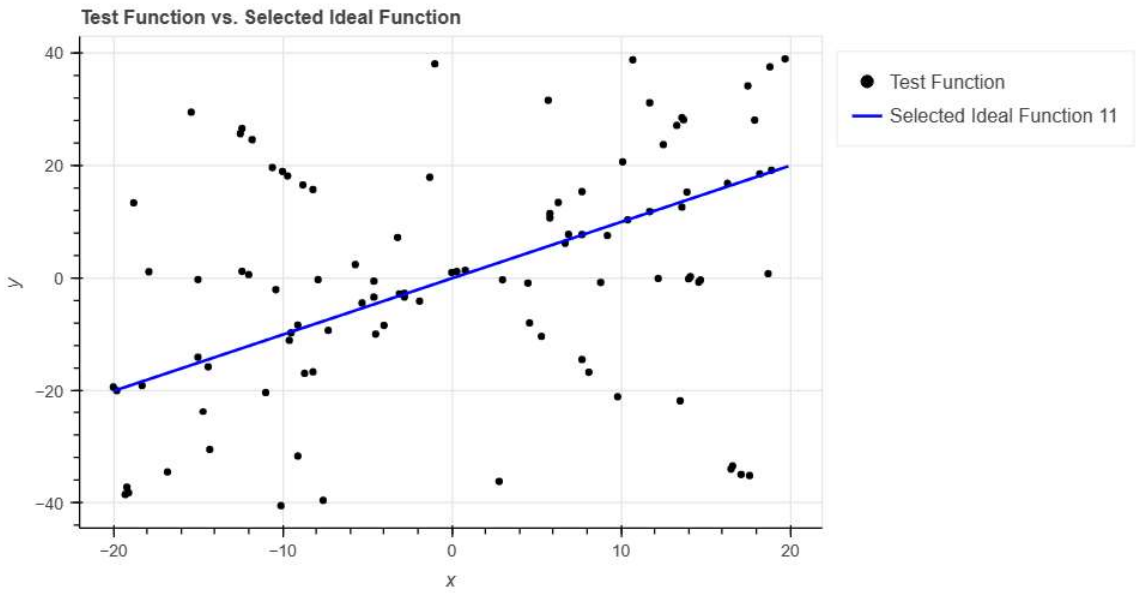

Figure 3 : Representation of selected function for test data

Github exercise :

```
# Clone the repository and checkout the develop branch
git clone <repository_url>
cd <repository_name>
git checkout develop

# Create and checkout a new branch for your work
git checkout -b feature/new-function

# Implement your new function and save the changes

# Stage your changes for commit
git add .

# Commit your changes with a descriptive message
git commit -m "Add new function for <description>"

# Push your changes to your fork of the repository
```

```
git push origin feature/new-function

# Go to the repository on GitHub and create a pull request
# Navigate to your fork of the repository
# Click on the "Pull requests" tab
# Click on the "New pull request" button
# Choose develop as the base branch and your feature/new-
function branch as the compare branch
# Add a descriptive title and description for your pull
request
# Click on the "Create pull request" button

# Wait for your team members to review your changes. They may
leave comments or approve the pull request.

# After approval, a team member with merge permissions can
merge your pull request into the develop branch.
# They can do this either through the GitHub interface or by
using the following command:
git checkout develop
git pull origin develop
git merge feature/new-function
git push origin develop
```
