PROGRAMMING WITH PYTHON: A DATA PIPELINE FOR IDEAL FUNCTION SELECTION, TEST MAPPING, AND VISUALIZATION

Hriday Dewan

10246379

Table of Contents

1. Introduction

1.1 Background and Rationale

1.1.1 Background

It has been underscored from the background research that the Python programming language became a coherent choice for solving data-driven problems, as well as being used in academic and professional contexts. The cogent support of the various libraries and versatility have made Python a great choice for researchers, data analysts, and software engineers who have been working with big datasets, as well as developing meaningful visualisations and statistical or mathematical models for making accurate decisions. The research developed by McKinney (2017) shows that Python has coherent data-handling capabilities. Such as the use of pandas and numpy libraries, it is possible to clean the data, transform the data, and perform statistical analysis to deliver cogent insights. For this project, the developer will develop a Python-based solution with the help of a Jupyter Notebook environment to ingest data, as well as integrate with the database, and develop a visualisation with the bokeh library to show the relationships.

1.1.2 Rationale

The research shows the major challenge of data analysis is making datasets able to be processed in a consistent manner, stored safely and interpreted efficiently. The mapping of datasets in comparison with a reference or ideal dataset helps the analysts to evaluate fit, detect deviations and confirm the data quality. Also, the fact that the results are integrated into a database makes them long-term accessible and reproducible, and the visualisation layer of interpretation is intuitive and delivers support to technical and non-technical stakeholders. In academia, a pipeline of this type shows a knowledge of the end-to-end data science process, including ingestion through to communicating insights (Zhang et al., 2024). These skills are also applicable directly to data engineering, analytics, and applied research. In the IU module DLMDSPWP01 (Programming with Python), researchers engage in such an assignment to connect the abstract programming concepts with real-life, practical data problems. Although introductory courses can be based on syntax and the simplest forms of programming, this task forces students to integrate object-oriented design, database operations, algorithmic thinking, and visualisation. By so doing, the assignment reflects the competency set that should be integrated in the role of data practitioners and supports the academic goal of expressing technical ability and critical thinking.

1.2 Aim of the Project

The major aim of this project is to develop a Python-based pipeline, which ingests training, ideal, and test datasets as well as uses an algorithm to underscore ideal functions with the help of least-squares minimisation. The pipeline will then compare the test points to the selected functions and store the results in an SQLite database and develop an interactive

visualisation with Bokeh. In addition to the technical procedures, the assignment is also meant to describe the implementation of the software engineering concepts of modularity, object-oriented programming (OOP), exception handling, and unit testing. The implementation will mainly be developed with various kinds of Python libraries such as pandas, numpy, SQLAlchemy, and Bokeh. Every library plays a beneficial role in developing the project, such as pandas has been used to manipulate structured data, numpy is used to execute mathematical operations, SQLAlchemy is used to interact with a database, and Bokeh is used to visualise data. The combination of these tools into a workflow, the assignment shows not just a competence in Python coding but also an understanding of how to create a full-stack data solution.

1.3 Scope and Boundaries

The main agenda of this project is to handle the dataset, develop algorithmic mapping, visualisation, exception handling, and testing. All of these aspects meet the course objectives, and they present an adequate representation of programming and applied data analysis. The boundaries are also crucial: state-of-the-art machine learning can be used, distributed databases, including PostgreSQL clusters, or massive cloud implementations are beyond the scope of the project (Dritsas and Trigka, 2025). Also, these could convey future extensions; they are not needed to meet the present academic objectives.

1.4 Structure of the Report

The developer of this project uses a cogent structure to deliver clear and logical ideas. In the introduction section, the developer underscores the rationale, aims, and scope of the assignment, as well as in the main body, the developer critically analyses the literature, defines methodology, design choices, implementation process, and results. This contains the explanation of the algorithmic logic, database design and the visualisation results. Also, the conclusion will sum up the key findings, analyse how effective the solution was, and consider potential future directions. With this structure, the report fulfils both academic and logical criteria, as well as reflects the logical process of creating and assessing a technical solution.

2. Literature and Context

This section has been established to critically analyse the published journal articles and underscore major technological innovations of Python programming in analysis, mapping, and visualisation, as well as outline the gap.

2.1 Python in Scientific Computing

In scientific computing and data science, Python has dominated over the last ten years because of its versatility and its wide range of libraries. McKinney (2018) underscores the efficiency of pandas in maintaining the structured data, whereas Harris et al. (2020) show that NumPy is very efficient for array programming in high-performance computing environments. It is underscored by Virtanen et al. (2020) that the ongoing growth of SciPy, whereby the

libraries of Python are being employed as a more scientific toolbox, including optimisation, signal processing, and statistics. Also, Perkel (2019) states that Python is the “lingua franca of science” that allows cross-disciplinary collaboration and decreases the barrier to entry for a researcher. Recent research (Agbo et al., 2019) demonstrates the effectiveness of Python in education to learn computational thinking faster and develop transferable skills in graduates. In this assignment, the choice of Python conforms to the following trends: the solution is a logical pipeline made of a combination of several libraries that both emulate the academic best practices and industry norms.

2.2 Databases in Data Science

Reproducibility and integrity in data workflows are based on databases. Sarmiento et al. (2021) contrasted SQL and NoSQL systems and stated that it is important to note that relational databases are cogent for structured data in which consistency is essential. Hipp (2022) highlights the relevance of SQLite, especially in teaching and research of lightweight and reproducible applications. It is very portable due to its serverless architecture, which is one of the reasons why it was used in this assignment. The research shows the debates which are based on the problem of accessibility versus scalability. Hernandez and Colom (2019) state that the emergence of cloud-native and distributed systems does not diminish the importance of lightweight databases, which are needed to create prototypes and maintain reproducibility. Also, Dritsas and Trigka (2025) show the importance of SQL abstractions as a bridging pedagogical concept to learners as they transition to more elaborate big data ecosystems.

2.3 Visualisation as a Communication Tool

The research has shown that visualisation plays a coherent role in order to deliver accurate insights to various audiences, as well as helps to underscore dataset distribution and patterns. The initial framework of interactivity by Heer and Shneiderman (2012) has been developed in recent research that focuses on narrative and accessibility. Munzner (2025) states that visualisation is an analytical and communicative tool that assists users in confirming their hypotheses in addition to investigating unforeseen patterns. More recently, Knaflic (2020) stressed the role of storytelling in data visualisation and supported the need to communicate the results clearly and simply. Interactive visualisation also helps to deliver cogent insights to organisations and individuals. Borland et al. (2021) demonstrate how applications such as Bokeh and Plotly can convert static results into a more exploratory experience, which can be even more reproducible since the user can directly question the data. In this assignment, this is implemented through the visualisation of training data, selected ideal functions, and test mappings: this converts the abstract results of an algorithm into a narrative, which can be intuitively understood by stakeholders.

3. Methodology and Design

The developer of this project has decided to use a coherent methodology which aligns with the object-oriented programming (OOP) methods, pipeline construction, as well as cogent database integration. With the help of this methodological design, we can confirm technical correctness and in compliance with the academic criteria of reproducibility, transparency and critical reflection.

3.1 Programming Approach

Here, the Python programming language has been used with OOP to confirm modularity and reusability. The design started with a BaseDataset class that contained functionality for loading and verifying datasets. Also, the developer has been developing TrainingDataset, IdealFunctions, and TestDataset classes for managing the individual requirements of each input file. This type of inheritance shows the pedagogical and practical benefits of OOP, such as abstraction, encapsulation, and maintainability (Lutz, 2021). Also, the implementation process design is developed with a cognet pipeline whereby individual stages that included data ingestion, mapping, database storage and visualisation were modularised and implemented as separate components. This pipeline model is a representation of contemporary data science practice, in which modularity enables iterative development as well as research collaboration. Every element can be enhanced, tested, as well as replaced without interfering with the whole process.

3.2 Database Management

SQLite was used to deliver persistence of data, and it was interfaced by SQLAlchemy. SQLite has been chosen due to its portability as well as its simplicity and the ability to use it with educational settings (Hipp, 2022). SQLite also does not need external configuration, unlike server-based databases, and is therefore the best choice for reproducibility in an academic context. SQLAlchemy also delivered a layer of abstraction that not only makes interaction with the database easier but also makes the codebase flexible to other relational systems in case of need. It has been underscored from research that coherent storage made it possible to capture training, ideal and test datasets reproducibly, as well as the mapping results. This was a methodological decision such as the developer will store intermediate and final results in a relational database, and it confirmed consistency, audit-friendly, and was reflective of research practice in which traceable streams of data are critical (Gundersen and Kjensmo, 2018).

3.3 Mapping Logic

The major agenda of developing this analytical task is to match the training functions with their nearest ideal functions, as well as allocate test points to these mappings. The minimisation of least-squares in the mapping logic has been based on the computation of the sum of squared deviations in training and ideal values of a candidate function. The best match will be selected

as the function that had the least error. After developing the mappings, the developer will evaluate the test points to determine their divergence from the chosen ideal function. Tolerance threshold has been used, which is set as the maximum training deviation $\sqrt{2}$. This criterion is suggested in the project specification; the sensitivity and robustness are balanced as well as the points falling within the threshold are assigned with certainty, whereas the points exceeding it are marked as unassigned.

3.4 Exception Handling

In this project, the developer has been using custom exceptions such as DataFormatError, which is used for malformed inputs, and DatabaseError is used for storage problems. The use of explicit error handling confirms failures were reported in an open manner as opposed to resulting in silent or vague malfunctions. The practice is indicative of concepts of software engineering, in which resilience and clarity are coherent for system trustworthiness (Pressman and Maxim, 2020).

3.5 Unit Testing

In order to determine that the developed functionality is correct, we will use unit testing. Mainly, the assertion-based tests are to be used for verifying dataset loading, mapping logic, and database integration. This test-driven method gave the ability to be confident about the correctness and detect regressions early during the pipeline. Unit testing, in an academic setting, is associated with the principles of reproducibility and rigour as well as need to confirm that results are not only correct but demonstrably verifiable.

4. Implementation and Results

4.1 Implementation

4.1.1 Custom Exceptions and Dataset Classes

```
[2]
✓ 0s
TRAIN_CSV = 'train.csv'
IDEAL_CSV = 'ideal.csv'
TEST_CSV = 'test.csv'
DB_PATH = 'results.db'
EXPORT_MAPPING_CSV = 'test_mapping.csv'
BOKEH_HTML = 'visualization.html'
```

Import libraries

```
[3]
✓ 2s
import os
import math
import logging
import pandas as pd
import numpy as np
from typing import List, Tuple, Dict, Optional
from sqlalchemy import create_engine
from sqlalchemy.exc import SQLAlchemyError
from bokeh.plotting import figure, output_file, save
from bokeh.models import ColumnDataSource, HoverTool
logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')
```

Custom Exceptions

```
[4]
✓ 0s
class DataFormatError(Exception):
    """Raised when input CSV format is not as expected."""
    pass
class DatabaseError(Exception):
    """Raised for database-related errors."""
    pass
```

Figure 1: Define global paths, import libraries, and use custom exceptions

It has been underscored from the above figure (1) that the developer uses the Google Colab environment and defines file paths, importing necessary libraries, configuring logging, as well as developing custom exceptions. By developing these approaches, we can coherently manage data, reporting errors and reproducibility. This cogent pipeline boosts the accuracy of data analysis and visualisation.

```
[5]
✓ 0s
class BaseDataset:
    def __init__(self, path: str):
        self.path = path
        self.df: Optional[pd.DataFrame] = None

    def load(self):
        try:
            self.df = pd.read_csv(self.path)
        except Exception as e:
            raise DataFormatError(f'Could not read CSV at {self.path}: {e}')
        return self.df
class TrainingDataset(BaseDataset):
    """
    Training dataset; expects X and 4 Y columns (Y1..Y4).
    Demonstrates inheritance from BaseDataset.
    """
    def validate(self):
        if self.df is None:
            raise DataFormatError('Training data not loaded yet.')
        cols = list(self.df.columns)
        if len(cols) < 5:
            raise DataFormatError('Training CSV must contain at least 5 columns: X and 4 Ys.')
        # Ensure first column is X
        # We'll accept columns starting with X or x
        if cols[0].lower() != 'x':
            logging.warning('First column not named X — treating first column as X anyway.')

    def get_x(self) -> np.ndarray:
        return self.df.iloc[:, 0].values

    def get_y_columns(self) -> List[str]:
        return list(self.df.columns[1:5])

    def get_training_series(self) -> Dict[str, pd.Series]:
        """Return mapping of training column name -> series"""
        cols = self.get_y_columns()
        return {c: self.df[c] for c in cols}
```

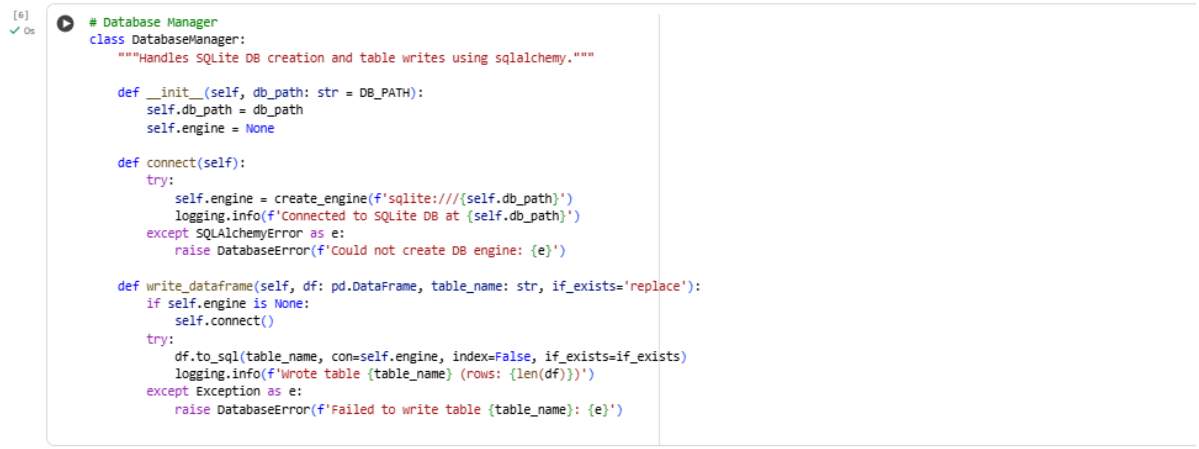
Figure 2: Base Dataset class and developing a training dataset class from BaseDataset

```python
class IdealFunctions(BaseDataset):
    """Loads ideal functions CSV (X + 50 columns)."""
    def validate(self):
        if self.df is None:
            raise DataFormatError('Ideal functions not loaded yet.')
        if self.df.shape[1] < 2:
            raise DataFormatError('Ideal CSV must contain X and at least one ideal function column.')

    def get_x(self) -> np.ndarray:
        return self.df.iloc[:, 0].values

    def get_function_columns(self) -> List[str]:
        return list(self.df.columns[1:])

    def get_function_series(self, name: str) -> pd.Series:
        return self.df[name]


class TestDataset(BaseDataset):
    """Loads test CSV with X,Y pairs."""
    def validate(self):
        if self.df is None:
            raise DataFormatError('Test data not loaded yet.')
        if self.df.shape[1] < 2:
            raise DataFormatError('Test CSV must contain X and Y columns.')

    def get_pairs(self) -> List[Tuple[float, float]]:
        xs = self.df.iloc[:, 0].values
        ys = self.df.iloc[:, 1].values
        return list(zip(xs, ys))
```

Figure 3: Ideal function class and Test Dataset class

The above figure (2 and 3) shows that the developer creates various kinds of classes to handle the object-oriented structure of input data. The BaseDataset class has been delivered functionality for loading CSV files into pandas DataFrames as well and custom exceptions are used to make sure that errors are identified. TrainingDataset has been validating that the files have an X column and four Y columns, such as delivering cogent methods to retrieve X values and Y-series. Also, IdealFunctions confirms that X exists and there are various columns of functions, which allow the retrieval of the data of functions. Testdataset checks X–Y pairs and extracts them in the form of tuples. The combination guarantees strong, reusable, and modular manipulation of datasets, which can facilitate reproducibility and simplify cogent analysis.

4.1.2 Database Manager

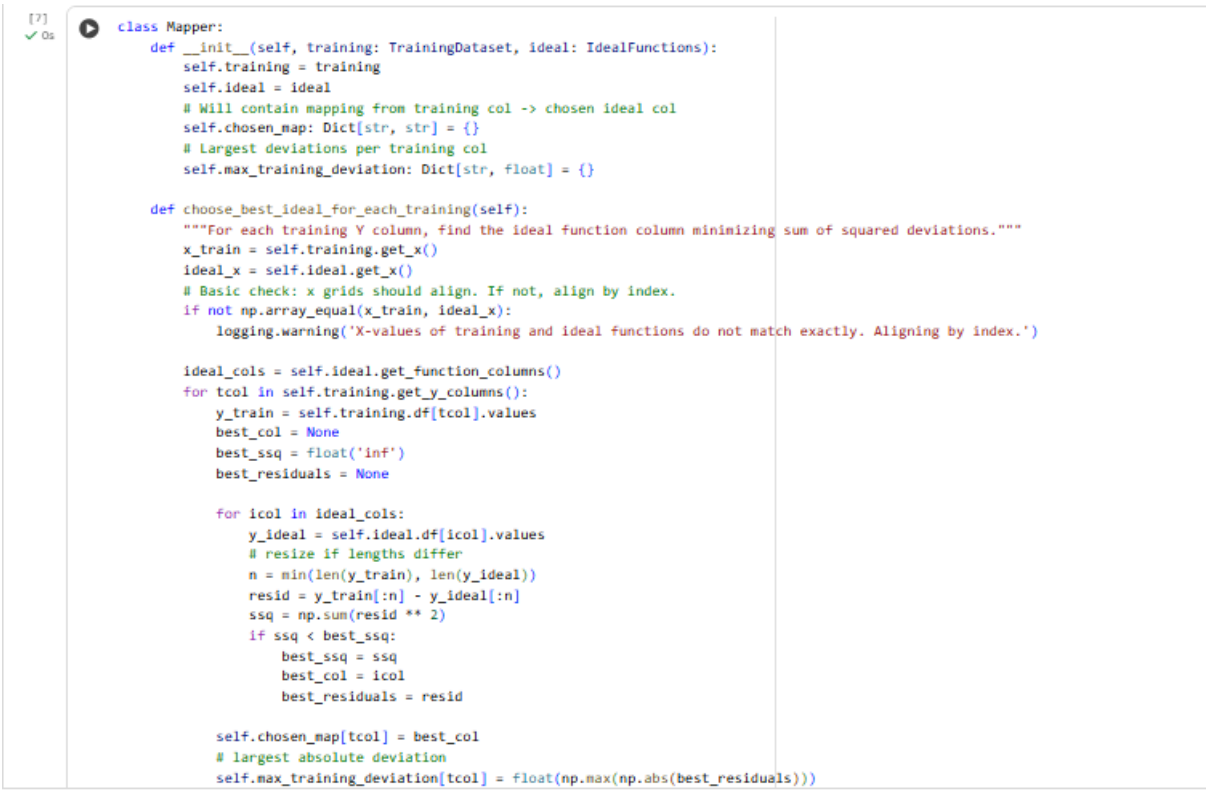


```
[6]
✓ 0s
# Database Manager
class DatabaseManager:
    """Handles SQLite DB creation and table writes using sqlalchemy."""

    def __init__(self, db_path: str = DB_PATH):
        self.db_path = db_path
        self.engine = None

    def connect(self):
        try:
            self.engine = create_engine(f'sqlite:///{self.db_path}')
            logging.info(f'Connected to SQLite DB at {self.db_path}')
        except SQLAlchemyError as e:
            raise DatabaseError(f'Could not create DB engine: {e}')

    def write_dataframe(self, df: pd.DataFrame, table_name: str, if_exists='replace'):
        if self.engine is None:
            self.connect()
        try:
            df.to_sql(table_name, con=self.engine, index=False, if_exists=if_exists)
            logging.info(f'Wrote table {table_name} (rows: {len(df)})')
        except Exception as e:
            raise DatabaseError(f'Failed to write table {table_name}: {e}')
```

Figure 4: Database Manager

Figure 4 shows the developer developing a DatabaseManager class, which helps to handle database operations with the help of SQLite and SQLAlchemy, and confirms structured storage of results. It has helped to develop connections to the database, log activity and write pandas DataFrames to tables with the ability to replace or append. It eases the persistence of data, ensures reproducibility, and offers reliability due to error handling because of the ability to abstract database tasks. This design will provide security of raw and processed data, traceability and easy retrieval of both data to conduct analysis and validation.

4.1.3 Mapping Logic

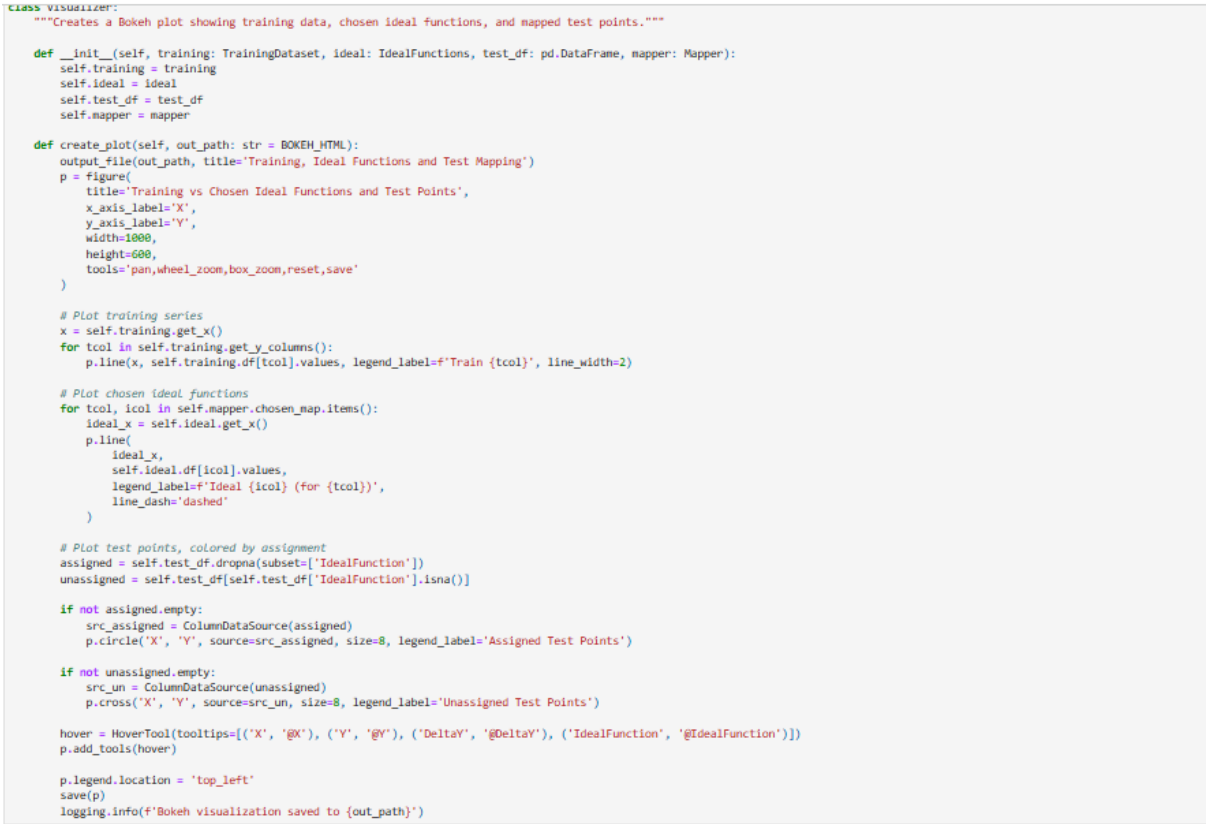


```
[7]
✓ 0s
class Mapper:
    def __init__(self, training: TrainingDataset, ideal: IdealFunctions):
        self.training = training
        self.ideal = ideal
        # Will contain mapping from training col -> chosen ideal col
        self.chosen_map: Dict[str, str] = {}
        # Largest deviations per training col
        self.max_training_deviation: Dict[str, float] = {}

    def choose_best_ideal_for_each_training(self):
        """For each training Y column, find the ideal function column minimizing sum of squared deviations."""
        x_train = self.training.get_x()
        ideal_x = self.ideal.get_x()
        # Basic check: x grids should align. If not, align by index.
        if not np.array_equal(x_train, ideal_x):
            logging.warning('X-values of training and ideal functions do not match exactly. Aligning by index.')

        ideal_cols = self.ideal.get_function_columns()
        for tcol in self.training.get_y_columns():
            y_train = self.training.df[tcol].values
            best_col = None
            best_ssq = float('inf')
            best_residuals = None

            for icol in ideal_cols:
                y_ideal = self.ideal.df[icol].values
                # resize if lengths differ
                n = min(len(y_train), len(y_ideal))
                resid = y_train[:n] - y_ideal[:n]
                ssq = np.sum(resid ** 2)
                if ssq < best_ssq:
                    best_ssq = ssq
                    best_col = icol
                    best_residuals = resid

            self.chosen_map[tcol] = best_col
            # largest absolute deviation
            self.max_training_deviation[tcol] = float(np.max(np.abs(best_residuals)))
```

Figure 5: Core Mapping Logic

It has been underscored from the above figure (5) that the developer develops a Mapper class, which helps to develop the cogent analytical pipeline and link training data with ideal functions via least-squares minimisation. Mainly, it helps to choose the nearest similar ideal function to each training column, and it records the highest deviations. The test points are then compared to these mappings through interpolation and a $\sqrt{2}$ deviation threshold. This guarantees that acceptable tolerance points are assigned, maintaining mathematical rigour, accuracy, and strength in mapping results and rejection of inconsistent data.

4.1.4 Visualisation logic



```
class Visualizer:
    """Creates a Bokeh plot showing training data, chosen ideal functions, and mapped test points."""

    def __init__(self, training: TrainingDataset, ideal: IdealFunctions, test_df: pd.DataFrame, mapper: Mapper):
        self.training = training
        self.ideal = ideal
        self.test_df = test_df
        self.mapper = mapper

    def create_plot(self, out_path: str = BOKEH_HTML):
        output_file(out_path, title='Training, Ideal Functions and Test Mapping')
        p = figure(
            title='Training vs Chosen Ideal Functions and Test Points',
            x_axis_label='X',
            y_axis_label='Y',
            width=1000,
            height=600,
            tools='pan,wheel_zoom,box_zoom,reset,save'
        )

        # Plot training series
        x = self.training.get_x()
        for tcol in self.training.get_y_columns():
            p.line(x, self.training.df[tcol].values, legend_label=f'Train {tcol}', line_width=2)

        # Plot chosen ideal functions
        for tcol, icol in self.mapper.chosen_map.items():
            ideal_x = self.ideal.get_x()
            p.line(
                ideal_x,
                self.ideal.df[icol].values,
                legend_label=f'Ideal {icol} (for {tcol})',
                line_dash='dashed'
            )

        # Plot test points, colored by assignment
        assigned = self.test_df.dropna(subset=['IdealFunction'])
        unassigned = self.test_df[self.test_df['IdealFunction'].isna()]

        if not assigned.empty:
            src_assigned = ColumnDataSource(assigned)
            p.circle('X', 'Y', source=src_assigned, size=8, legend_label='Assigned Test Points')

        if not unassigned.empty:
            src_un = ColumnDataSource(unassigned)
            p.cross('X', 'Y', source=src_un, size=8, legend_label='Unassigned Test Points')

        hover = HoverTool(tooltips=[('X', '@X'), ('Y', '@Y'), ('DeltaY', '@DeltaY'), ('IdealFunction', '@IdealFunction')])
        p.add_tools(hover)

        p.legend.location = 'top_left'
        save(p)
        logging.info(f'Bokeh visualization saved to {out_path}')
```

Figure 6: Visualisation class and logic

The above figure (6) implies that the Visualizer class has helped to generate Bokeh plots to show training data, matched ideal functions, and test point mappings. It graphically shows the training series as solid lines, ideal functions as dashed lines and test points as circles, as well as crosses depending on the assignment status. Hover tools give in-depth insights into values, deviations, and function assignments. This visualisation has helped to eliminate computation and interpretation, accelerate transparency, reproducibility, and accessibility by converting numerical results into an intuitive, interactive narrative for stakeholders.

4.1.5 Unit Testing

```python
def run_pipeline(train_csv: str = TRAIN_CSV, ideal_csv: str = IDEAL_CSV, test_csv: str = TEST_CSV):
    # Load training data
    training = TrainingDataset(train_csv)
    training.load()
    training.validate()

    # Load ideal functions
    ideal = IdealFunctions(ideal_csv)
    ideal.load()
    ideal.validate()

    # Load test data
    test = TestDataset(test_csv)
    test.load()
    test.validate()

    # DB manager
    db = DatabaseManager()
    db.connect()
    # write raw tables
    db.write_dataframe(training.df, 'training')
    db.write_dataframe(ideal.df, 'ideal_functions')
    db.write_dataframe(test.df, 'test_raw')

    # Mapper
    mapper = Mapper(training, ideal)
    mapper.choose_best_ideal_for_each_training()
    test_pairs = test.get_pairs()
    mapping_df = mapper.map_test_points(test_pairs)

    # Save mapping to DB and CSV
    db.write_dataframe(mapping_df, 'test_mapping')
    mapping_df.to_csv(EXPORT_MAPPING_CSV, index=False)
    logging.info(f'Mapping exported to {EXPORT_MAPPING_CSV}')

    # Visualization
    viz = Visualizer(training, ideal, mapping_df, mapper)
    viz.create_plot(BOKEH_HTML)

    return {
        'db_path': db.db_path,
        'mapping_csv': EXPORT_MAPPING_CSV,
        'bokeh_html': BOKEH_HTML,
    }
```

Figure 7: Utilities and Unit Tests

```
[10]
✓ 0s

def _unit_tests():
    logging.info('Running basic unit tests...')

    #  Check files exist
    for p in [TRAIN_CSV, IDEAL_CSV, TEST_CSV]:
        assert os.path.exists(p), f'Expected file not found: {p}'

    #  Load and basic checks
    tr = TrainingDataset(TRAIN_CSV)
    tr.load()
    tr.validate()

    idf = IdealFunctions(IDEAL_CSV)
    idf.load()
    idf.validate()

    te = TestDataset(TEST_CSV)
    te.load()
    te.validate()

    #  Mapper choose logic
    m = Mapper(tr, idf)
    chosen = m.choose_best_ideal_for_each_training()
    assert len(chosen) >= 1, 'No chosen mappings found.'

    # Map test points
    mapping_df = m.map_test_points(te.get_pairs())
    assert 'IdealFunction' in mapping_df.columns, 'Mapping result missing IdealFunction column.'

    logging.info('Unit tests passed (basic).')
```

Figure 8: Unit tests using assertions

It has been underscored from the above figure (7 and 8) that the developer develops a _unit_tests() function to confirm the correctness, reliability and reproducibility of the pipeline through assertion-based tests on all the stages. It initially checks the availability of the necessary files of the dataset, which protects against the missing of inputs. Then, we perform data loading and validation of training data, ideal data, and test data, and also confirm structural integrity. The function also confirms that the Mapper has been able to find at least one ideal function for a training series. Also, it validates the test mapping results, which prove that the necessary IdealFunction column is created. All these tests give the confidence that the system works and is cogent, as well as stating coherent software engineering and academic best practices.

4.2 Results

This section is established to show the output of the implementation process. The coding outputs will confirm the software correctness and validity.

```python
if __name__ == '__main__':
    try:
        _unit_tests()
        results = run_pipeline()
        logging.info('Pipeline completed successfully. Outputs:')
        for k, v in results.items():
            logging.info(f' - {k}: {v}')
    except AssertionError as ae:
        logging.error(f'AssertionError during tests/pipeline: {ae}')
    except DataFormatError as dfe:
        logging.error(f'Data format problem: {dfe}')
    except DatabaseError as dbe:
        logging.error(f'Database problem: {dbe}')
    except Exception as e:
        logging.exception(f'Unhandled exception: {e}')
```

Figure 9: Running the Pipeline

The aforementioned coding operation served as a program's entry point, where we develop unit tests and run the pipeline. The pipeline has generated a test mapping csv file, a database file, as well as a Bokeh plot in HTML format, which are described in the section below.

X	Y	DeltaY	IdealFunction
17.5	34.16104	0.351148	y41
0.3	1.215102	0.467342	y41
-8.7	-16.8439		
-19.2	-37.1709		
-11	-20.2631		
0.8	1.426456	0.532222	y41
14	-0.06651	0.134233	y48
-10.4	-2.00709		
-15	-0.20536	0.452371	y48
5.8	10.71137	0.656326	y41
-7.6	-39.4954		
-19.8	-19.915	0.115014	y11
18.9	19.19325	0.293245	y11
6.9	7.801455		
8.8	-0.72605	0.48884	y48
-9.5	-9.65225	0.152251	y11
8.1	-16.6595	0.337686	y42
7.7	-14.4		
4.5	-0.84011		
10.7	38.79003		

Figure 10: Output of mapping

It can be underscored from the test mapping that unseen test points align with the chosen ideal functions, which have been derived from training data. Also, every record shows the test point's X and Y values as well as the difference from the matched function and whether it falls within the acceptable threshold. The mapping logic is validated by assigned points, and outliers are shown by unassigned points. This delivers accuracy, transparency, and robustness, which shows that the algorithm is trustworthy and cautious with inconsistent data.

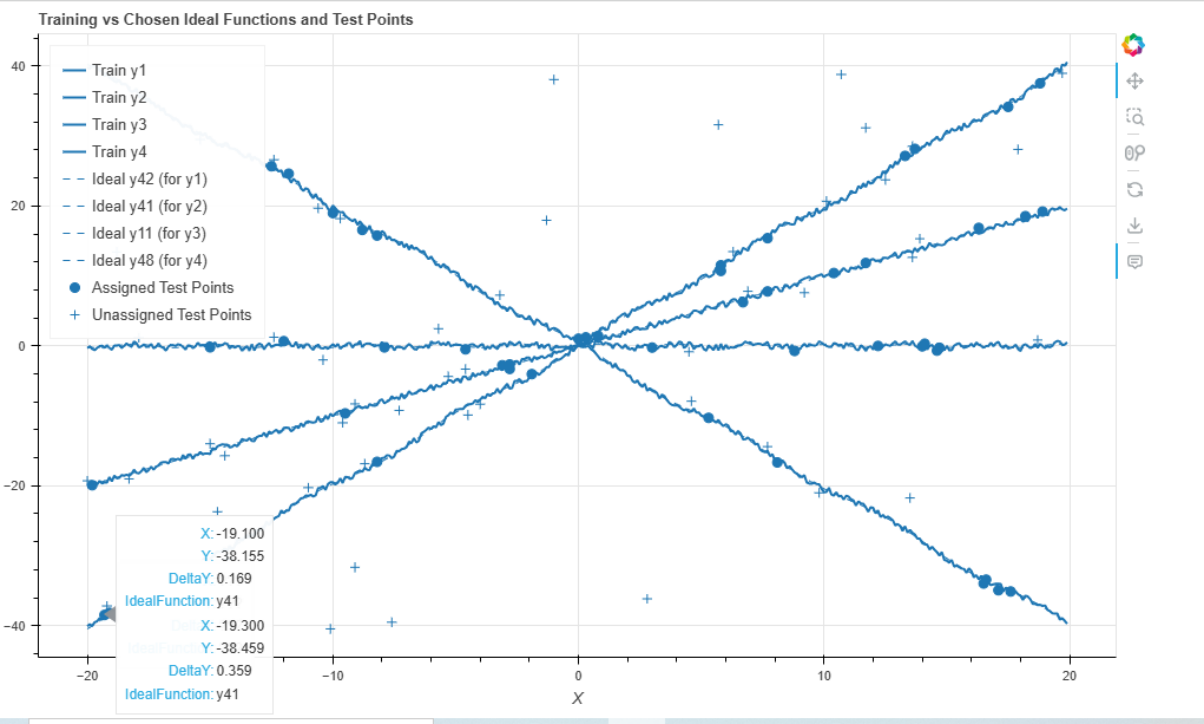


Figure 11: Training vs chosen ideal functions and test points

It has been determined from the above Bokeh visualisation that the data pipeline correctly matched four training datasets, such as y1 through y4, with their corresponding ideal functions, such as Ideal y42 for y1. The training functions we show as solid lines, as well as corresponding ideal functions have been shown as dashed lines, and test points as markers. The figure (11) also shows that the assigned test points are within the expected deviation of these ideal functions, whereas the unassigned test points are viewed as outliers. This Bokeh visualisation has helped to show the accuracy and the strength of the mapping algorithm.

```
[12]
✓ 0s
```

```
import sqlite3
import pandas as pd

# Connect to the DB
conn = sqlite3.connect("results.db")

# Check which tables exist
tables = pd.read_sql("SELECT name FROM sqlite_master WHERE type='table';", conn)
print("Tables in DB:")
print(tables)
```

```
Tables in DB:
              name
0         training
1  ideal_functions
2         test_raw
3     test_mapping
```

Figure 12: Tables in the database

ideal_functions (400 rows) Export

```
SELECT * FROM 'ideal_functions' LIMIT 0,30
```

Execute

x	y1	y2	y3	y4	y5	y6	y7	y8	y9	y10	y11	y12	y13	y14	y15	y16
-20	-0.9129453	0.40808207	9.087055	5.408082	-9.087055	0.9129453	-0.8390715	-0.85091937	0.81616414	18.258905	-20	-58	-45	20	13	400
-19.9	-0.8676441	0.4971858	9.132356	5.4971857	-9.132356	0.8676441	-0.8652126	0.16851768	0.9943716	17.266117	-19.9	-57.7	-44.8	19.9	12.95	396.01
-19.8	-0.81367373	0.58132184	9.186326	5.5813217	-9.186326	0.81367373	-0.88919115	0.6123911	1.1626437	16.11074	-19.8	-57.4	-44.6	19.8	12.9	392.04
-19.7	-0.75157344	0.65964943	9.248426	5.6596494	-9.248426	0.75157344	-0.91094714	-0.99466854	1.3192989	14.805996	-19.7	-57.1	-44.4	19.7	12.85	388.09
-19.6	-0.6819636	0.7313861	9.318036	5.731386	-9.318036	0.6819636	-0.9304263	0.7743557	1.4627723	13.366487	-19.6	-56.8	-44.2	19.6	12.8	384.16
-19.5	-0.60553986	0.795815	9.39446	5.795815	-9.39446	0.60553986	-0.9475798	-0.11702018	1.59163	11.808027	-19.5	-56.5	-44	19.5	12.75	380.25
-19.4	-0.52306575	0.8522923	9.476934	5.8522925	-9.476934	0.52306575	-0.96236485	-0.59004813	1.7045846	10.147476	-19.4	-56.2	-43.8	19.4	12.7	376.36
-19.3	-0.43536535	0.90025383	9.564634	5.900254	-9.564634	0.43536535	-0.97474456	0.97776526	1.8005077	8.402552	-19.3	-55.9	-43.6	19.3	12.65	372.49
-19.2	-0.34331492	0.93922037	9.656685	5.9392204	-9.656685	0.34331492	-0.98468786	-0.87895167	1.8784407	6.5916467	-19.2	-55.6	-43.4	19.2	12.6	368.64
-19.1	-0.2478342	0.96880245	9.752166	5.9688025	-9.752166	0.2478342	-0.99217	0.37579286	1.9376049	4.7336335	-19.1	-55.3	-43.2	19.1	12.55	364.81
-19	-0.1498772	0.9887046	9.850122	5.9887047	-9.850122	0.1498772	-0.9971722	0.27938655	1.9774092	2.847667	-19	-55	-43	19	12.5	361
-18.9	-0.050422687	0.998728	9.949577	5.998728	-9.949577	0.050422687	-0.99968195	-0.80255306	1.997456	0.9529888	-18.9	-54.7	-42.8	18.9	12.45	357.21
-18.8	0.04953564	0.9987724	10.049536	5.998772	-10.049536	-0.04953564	-0.99969304	0.9999414	1.9975448	-0.93127006	-18.8	-54.4	-42.6	18.8	12.4	353.44
-18.7	0.14899902	0.98883736	10.148999	5.9888372	-10.148999	-0.14		-0.82669914	1.9776747	-2.7862818	-18.7	-54.1	-42.4	18.7	12.35	349.69
-18.6	0.24697366	0.9690222	10.246974	5.9690223	-10.246974	-0.24		0.3753813	1.9380444	-4.59371	-18.6	-53.8	-42.2	18.6	12.3	345.96

1 / 14

Figure 13: Database view from the SQL viewer (ideal_functions)

ideal_functions (400 rows) Export

```
SELECT * FROM 'test_mapping' LIMIT 60,30
```

Execute

X	Y	DeltaY	IdealFunction
10.4	10.410377	0.01037700000000008	y11
-19.1	-38.155376	0.16854100000000471	y41
-14.7	-23.71317		
9.2	7.5970726		
-9.6	-10.995601		
-3.2	7.2523284		
-10.6	19.67348		
-12.5	25.675346	0.1764460000000021	y42
-5.3	-4.3722925		
14.6	-0.677509	0.47775363000000004	y48
-16.8	-34.480774		
2.842171e-13	1.0261426	0.5261426	y42
5.7	31.600712		
-4.6	-0.4953994	0.43	y48
-15	-13.989205		

3 / 4

Figure 14: Database view from the SQL viewer (test_mapping)

```
SELECT * FROM 'test_raw' LIMIT 60,30
```

Execute

x	y
10.4	10.410377
-19.1	-38.155376
-14.7	-23.71317
9.2	7.5970726
-9.6	-10.995601
-3.2	7.2523284
-10.6	19.67348
-12.5	25.675346
-5.3	-4.3722925
14.6	-0.677509
-16.8	-34.480774
2.842171e-13	1.0261426

Figure 15: Database view from the SQL viewer (test_raw)

It has been underscored from the figure (12, 13,14 and 15) that the developer shows the database output via executing queries. The database output shows that raw and processed data have been accurately stored in structured tables, as well as confirming transparency and reproducibility. The ***"training"*** table retains the original training data, making a reference to baseline data. The ***"ideal_functions"*** table has X-values as well as fifty candidate functions, which makes it possible to compare and map them accurately. The ***"test_raw"*** table contains unaltered X-Y test points and is the basis of the assignment decision. Also, it has been seen from the figure (14) that the ***"test_mapping"*** table determines test points and calculates deviations and the fact of whether they were confidently assigned to an ideal function or not. All of these outputs testify to the pipeline working well, provide traceability, reproducibility, and a strict connection of raw data to final mapping outputs.

ideal_functions (400 rows) Export

```
SELECT * FROM 'training' LIMIT 60,30
```

Execute

x	y1	y2	y3	y4
-14	28.152843	-28.66428	-14.1337805	-0.27583477
-13.9	27.976458	-27.830526	-14.347666	-0.048467655
-13.8	27.49858	-27.606556	-13.959812	-0.06256741
-13.7	27.319183	-27.805117	-13.305638	-0.6818926
-13.6	26.959885	-27.46691	-13.460358	-0.35765544
-13.5	27.010479	-27.046322	-13.545402	-0.050601155
-13.4	27.444553	-27.415642	-13.1895275	-0.6999692
-13.3	26.477736	-26.918652	-13.590001	0.22568727
-13.2	27.011068	-27.134518	-13.620864	-0.65312505
-13.1	26.715807	-26.272911	-13.044477	0.031488232
-13	26.56426	-26.450706	-12.866318	0.30753955
-12.9	25.957773	-26.156477	-12.878518	0.1487716
-12.8	25.722597	-25.995968	[illegible]	-0.492448
-12.7	25.730907	-25.139296	[illegible]	0.051321585

3 / 14

Figure 16: Database view from the SQL viewer (training)

The above figure (16) shows a training table that delivers the baseline dataset with X-values and four Y-columns (y1–y4), which can be used as benchmarks to carry out the least-squares

comparison. It captures the variability of the series, which is precise to align with the ideal functions. Also, it has helped to preserve reproducibility, accuracy and accurate evaluation of test points, which is crucial in the process of validating mappings.

5. Conclusion and Future Work

It can be concluded that the developer of this project accurately developed a Python-based data pipeline which can ingested training, ideal, and test datasets. Also, used least-squares mapping as well as stored results in SQLite, and use Bokeh to visualised. The use of OOP design, database management, exception handling, and unit testing showed technical correctness and scholarly sound. Findings were valid in mapping training data to ideal functions and valid test point assignments. In the future, it would be possible to expand this pipeline by using more advanced machine learning models that would to boost predictive accuracy, scaling to larger relational or distributed databases, and cloud-based visualization dashboards to improve accessibility. Such extensions would enhance the relevance in the real world and retain the reproducibility and transparency in this study.

References

Zhang, C., Jia, L., Zhang, W. and Wen, N., 2024. Functional Programming Paradigm of Python for Scientific Computation Pipeline Integration. *arXiv preprint arXiv:2405.16956*. https://arxiv.org/abs/2405.16956

Lavanya, A., Gaurav, L., Sindhuja, S., Seam, H., Joydeep, M., Uppalapati, V., Ali, W. and SD, V.S., 2023. Assessing the performance of Python data visualization libraries: a review. *Int. J. Comput. Eng. Res. Trends*, *10*(1), pp.28-39. https://www.researchgate.net/profile/Lavanya-Addepalli/publication/369533034_Assessing_the_Performance_of_Python_Data_Visualization_Libraries_A_Review/links/6420a158315dfb4cceaee503/Assessing-the-Performance-of-Python-Data-Visualization-Libraries-A-Review.pdf

McKinney, W., 2012. *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. " O'Reilly Media, Inc.". https://books.google.com/books?hl=en&lr=&id=v3n4_AK8vu0C&oi=fnd&pg=PR3&dq=McKinney,+W.+(2018).+Python+for+data+analysis:+Data+wrangling+with+pandas,+NumPy,+and+IPython+(2nd+ed.).+O%E2%80%99Reilly+Media.&ots=rilLamArmv&sig=LGIhGHpHyyiEDNOEvhRYNi1Q248

Harris, C.R., Millman, K.J., Van Der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J. and Kern, R., 2020. Array programming with NumPy. *nature*, *585*(7825), pp.357-362. https://www.nature.com/articles/s41586-020-2649-2

Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J. and Van Der Walt, S.J., 2020. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature methods*, *17*(3), pp.261-272. https://www.nature.com/articles/s41592-019-0686-2

Perkel, J.M., 2018. Why Jupyter is data scientists' computational notebook of choice. *Nature*, *563*(7732), pp.145-147. https://go.gale.com/ps/i.do?id=GALE%7CA573082717&sid=googleScholar&v=2.1&it=r&linkaccess=abs&issn=00280836&p=HRCA&sw=w

Agbo, F.J., Oyelere, S.S., Suhonen, J. and Adewumi, S., 2019, November. A systematic review of computational thinking approach for programming education in higher education institutions. In *Proceedings of the 19th Koli calling international conference on computing education research* (pp. 1-10). https://dl.acm.org/doi/abs/10.1145/3364510.3364521

Sarmiento, D.E., Lebre, A., Nussbaum, L. and Chari, A., 2021. Decentralized SDN control plane for a distributed cloud-edge infrastructure: A survey. *IEEE Communications Surveys & Tutorials*, *23*(1), pp.256-281. https://ieeexplore.ieee.org/abstract/document/9319748/

Hipp, D. R. (2022). *SQLite documentation and architecture overview*. SQLite.org. Retrieved from https://sqlite.org

Hernandez, J.A. and Colom, M., 2025. Reproducible research policies and software/data management in scientific computing journals: a survey, discussion, and perspectives. *Frontiers in Computer Science*, *6*, p.1491823. https://www.frontiersin.org/journals/computer-science/articles/10.3389/fcomp.2024.1491823/full

Dritsas, E. and Trigka, M., 2025. A Survey on Database Systems in the Big Data Era: Architectures, Performance, and Open Challenges. *IEEE Access*. https://ieeexplore.ieee.org/abstract/document/11008610/

Gundersen, O.E. and Kjensmo, S., 2018, April. State of the art: Reproducibility in artificial intelligence. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 32, No. 1). https://ojs.aaai.org/index.php/AAAI/article/view/11503

Heer, J. and Shneiderman, B., 2012. Interactive dynamics for visual analysis: A taxonomy of tools that support the fluent and flexible use of visualizations. *Queue*, *10*(2), pp.30-55. https://dl.acm.org/doi/abs/10.1145/2133416.2146416

Munzner, T., 2025, August. Visualization analysis and design. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Courses* (pp. 1-2).https://dl.acm.org/doi/abs/10.1145/3721241.3733989

Knaflic, C.N., 2019. *Storytelling with data: let's practice!*. John Wiley & Sons. https://books.google.com/books?hl=en&lr=&id=aGatDwAAQBAJ&oi=fnd&pg=PR7&dq=Knaflic,+C.+N.+(2020).+Storytelling+with+data:+Let%E2%80%99s+practice!+Wiley.&ots=2DyaYIGWsU&sig=i8321iAY2vFlax93R09gDNkAj2M

Schmidt, J., 2020, February. Usage of Visualization Techniques in Data Science Workflows. In *VISIGRAPP (3: IVAPP)* (pp. 309-316). https://www.scitepress.org/PublishedPapers/2020/91819/91819.pdf

Lutz, M., 2014. *Python pocket reference*. " O'Reilly Media, Inc.". https://books.google.com/books?hl=en&lr=&id=5fujAgAAQBAJ&oi=fnd&pg=PP1&dq=Lutz,+M.+(2021).+Learning+Python,+5th+Edition.+O%E2%80%99Reilly+Media.&ots=w2Lbb7dKXJ&sig=CVAbrzzlgBXrLZs_1w0yHkaX5nY

Pressman, R.S., 2005. *Software engineering: a practitioner's approach*. Palgrave macmillan. https://books.google.com/books?hl=en&lr=&id=bL7QZHtWvaUC&oi=fnd&pg=PA1&dq=Pressman,+R.+S.,+%26+Maxim,+B.+R.+(2020).+Software+Engineering:+A+Practitioner%E2%80%99s+Approach,+9th+Edition.+McGraw-Hill.&ots=O9yf4MzR5m&sig=H8JEIkDDlaP1vQQfueaOCNAJPkg

Appendix 1: Collab Link

https://colab.research.google.com/drive/1R3mP9FUVlD84G3BuMyQxGDku0-ueiLGq?usp=sharing