TOWARDS USER CENTRIC AUTOMATION OF EVERYTHING

Semester: 3rd

MASTER OF TECHNOLOGY

By

KESHAN

Department of Computer Science & Engineering and Information Technology

JAYPEE INSTITUE OF INFORMATION TECHNOLOGY
(Declared Deemed to be University U/S 3 of UGC Act)
A-10, SECTOR-62, NOIDA, INDIA

December 2022

TABLE OF CONTENTS

DECLARATION BY THE SCHOLAR

I hereby declare that the work reported in the M.Tech Dissertation entitled **"Towards User Centric Automation of Everything"** submitted at **Jaypee Institute of Information Technology, Noida, India** is an authentic record of my work carried out under the supervision of **Prof Vikas Saxena**. I have not submitted this work elsewhere for any other degree or diploma. I am fully responsible for the contents of my M.Tech theses.

Name:

Department of Computer Science & Engineering and Information Technology
Jaypee Institute of Information Technology, Noida, India

Date:

SUPERVISOR'S CERTIFICATE

This is to certify that the work reported in the M.Tech Dissertation entitled **"Towards User Centric Automation of Everything"** submitted by **KESHAN** at **Jaypee Institute of Information Technology, Noida, India**, is a bonafide record of his original work carried out under my supervision. This work has not been submitted elsewhere for any other degree or diploma.

Name:

Affiliation:

Date:

PREFACE AND ACKNOWLEDGEMENT

I acknowledge the role of my supervisor Prof Vikas Saxena to guide me and give me enough freedom to explore in my ways.

ABSTRACT

Current automation technologies are expert centric and industry centric that benefit the few from the fruits of others' labor. Some initiatives are being taken to let any illiterate user give inputs in the form of touch, image, audio, video, text etc but so far these have not succeeded to generate the required quantum algorithm according to the need of user data. I have succeeded to conceive a new holistic automaton pipeline that will accept data from anyone and will create the most appropriate quantum algorithm to process the data whose outcomes can be visually presented. This new pipeline focuses on automated data collection, automated data cleaning, automated data modeling, automated functional optimization, data driven automated generation of quantum algorithm, automated hardware selection, automated process termination or the beginning of the next process. This will make everything user centric such that even those who have no formal education will also give their inputs and will watch the outcomes visually. This research is just a beginning of a new era that requires a better understanding of automated data modeling besides enriching the framework for automatically generating more and more complex quantum algorithms.

LIST OF FIGURES

LIST OF TABLES

1. Introduction: Understanding the Evolution of Automation of Everything

Since the beginning of human life, the ancestors focused on understanding their behavior and surroundings to make a better sense. This also led to imagine and envision about the intelligent machines. The Hindu mythology is full of science fiction stories for creating marvelous automated structures by engineer God Vishwakarma and Sorceress Maya, flying bird models by Rishi Bhardwaj, artificial life and life without sexual processes by many rishis. Ramayan and Mahabharat both mention animated servants, giant robots, flying chariots and synthetic swans etc that was much before the beginning of other civilizations. It is said that automaton soldiers guarded Buddha's relics and obeyed Ashoka like today's robots. Recursive rules of Sanskrit by Panini tell the far sightedness of Indian thinkers.

Other civilizations also mention about artificial servants, autonomous killing machines, and surveillance system etc. By 3rd century BC there was frequent flow of ideas between Indian and Greek thinkers. A Greek traveler had observed automated servants and self-propelled carts in the court of an Indian king. The culmination of ideas from ancient civilizations encouraged the engineers to construct devices mentioned in the mythological stories. Engineers from India, China and Greece began making self-moving devices, animated machines and other automatons described in ancient mythologies. The Greeks created instructions to make musical automata, mechanical servants and automata powered by steam, water, air and mechanics.

Islamic thinkers learnt deeply from Indian and Greek literature that helped them to take leaps and bounds to create a kind of scientific explanation of everything and create superior automaton devices. Their automaton devices were extremely superior to previous devices and were mesmerizing the Europeans who dreamt to construct more advanced calculating devices. This led to the beginning of a new type of computation and automaton.

The foundations of scientific method, as we understand it today, were laid down by Islamic thinkers. The great founder of scientific method Roger Bacon was the student of Islamic thinkers who learnt and interpreted old knowledge in a new way paving the foundations for scientific

method. He applied the empirical method of great Islamic thinker Ibn al-Haytham to understand Aristotle whose logic is the foundation of modern computing. The evolution of scientific method radically transformed human life by mechanizing the world picture and paving way for large scale industrial revolution. It gave a new conceptual framework to quantize human knowledge and laid the foundations to create new predictable and verifiable ideas. This led to a new type of automation where concepts were predicting the outcomes and ways to control the machines in the large scale industries. The automated machines that we try to construct today are all possible due to scientific method.

Artificial Intelligence is a new movement that aims to automate everything. It's still grounded in scientific method but its applications are not restricted to industrial automation only rather new initiatives are being taken to automate every aspect of human life. Though it's good to create novel technologies to automate everything, yet the consequences of hyper-automation may be fatal also. Obviously we need new conceptual foundations for user centric automation of everything than remaining enslaved with the industry centric automation of everything.

2. A CRITICAL ANALYSIS OF CURRENT TECHNOLOGIES FOR AUTOMATION OF EVERYTHING

A. AUTOML – GOOGLE AUTOML, AMAZON SAGEMAKER, AZURE AUTOML, DATAROBOT, H2O DRIVERLESS AI

Automated Machine Learning offers techniques and procedures to make Machine Learning accessible to those who are not specialists in it, to increase its effectiveness, and to quicken the pace of Machine Learning research. Recently, machine learning (ML) has seen significant success, and an increasing variety of disciplines now use it. However, the following tasks are performed by human machine learning experts:

- Cleaning and Preprocessing the data
- Selection and Construction of relevant features

- Selecting an appropriate data model
- Hyper-parameter Optimization
- Evaluating the model
- Deploying the model

The complexity of these jobs is frequently beyond the capabilities of non-ML professionals, hence the quick development of machine learning applications has increased the demand for ready-to-use machine learning techniques that don't require expert knowledge and can be easily used.

B. OPENAI

Auto Completion through Prompts:

- Start with some prompt. An answer is generated.
- Using some adjectives, the resulting completion is changed.
- Show and tell the model our needs.
- Adding some examples to our prompt can enhance the patterns. Apart from prompt design another important setting is known as temperature.
- Submitting same prompt multiple times will result in identical completions because the temperature was set to 0. Submitting the same prompt with temperature 1.
- Temperature lets us control the confidence level of the model when making predictions. Low temperature will result in fewer risks and accurate & deterministic completions while high temperature will result in more diverse completions.

C. LAMBEQ

1. Sentence input

- Syntax-based model: DisCoCat

- Bag-of-words: Spiders readers
- Word-sequence models: Cups and stairs readers
- Tree readers

2. **Diagram rewriting**

3. **Parameterization:** Classical case and Quantum case

4. **Training:** Classical case and Quantum case

D. CLASSIQ

1. Circuit Synthesis: The task of building a quantum algorithm is challenging, and solutions based on the principle of designing at the gate level do not scale. Similarly, the scope of solutions based on merging already-existing building blocks is very constrained. The Classiq platform enables functional-level high-level descriptions of quantum algorithms and automatically creates a corresponding circuit. The qubit count and circuit depth are just two examples of the resources that can be allocated and optimized during the synthesis process.

Quantum functions are used in the Classiq platform to perform the fundamental description of quantum algorithms. There are numerous ways to implement a quantum function, each with unique characteristic, such as the number of qubits, the number of auxiliary qubits, the depth, the approximation level, etc. Any level of function refinement, from the purely abstract to the completely defined, may be defined by the user. To deliver a concrete solution that satisfies all user requirements, the synthesis engine fills in the details.

Additionally, using various design decisions like function implementations, location, uncompute techniques, qubit management, wirings, etc., a full quantum algorithm can be

achieved in a variety of ways. The platform filters the design space in light of the user needs and resource limitations to discover the optimum realization.

1.1 Defining Algorithms:

Quantum algorithms are constructed from functions. Functions are the building blocks that implement quantum logic and data flow.

Calling a Function: Specifying name parameters. State preparation function is a built in function.

Defining the Data Flow: Connecting the output of a function to an input of another function. The state preparation function has a single pre-defined output and QFT has a single pre-defined input. The Classiq platform avoids the explicit description of both the functions and just requires the connection to be defined. The handling of functions with numerous inputs and outputs is demonstrated in the example below. Register slicing makes it feasible to divide a single input/output into several parts.

1.2 Circuit Constraints:

Some constraints need to be passed to the generation while synthesizing a circuit. Following constraints may be passed to the circuit generator:

- Depth: maximum depth of the circuit
- Width: maximum width of the circuit
- Gate Count: maximum number of times a gate appears

```
{
    "constraints": {
    "max_width": 20,
```

```
                "max_depth": 100,
                "max_gate_count": {
                "CX": 10,
                "T": 20
            }
        },
        "logic_flow": []
}
```

Optimization Parameter: Circuit can be optimized according to some parameter while synthesizing a circuit.

```
{
        "constraints": {
        "max_depth": 20,
        "max_gate_count": {
        "CX": 10,
        "T": 20
        },
        "optimization_parameter": "width"
        },
        "logic_flow": []
}
```

1.3 Synthesis Preferences:

User can modify the preferences of the synthesis process textual which includes output formats, hardware settings and timeouts.

Output Formats: Multiple output formats can be chosen. Different output options are:

‘qasm’ – OpenQASM;
‘qs’ – Q#;
‘ll’ – Microsoft’s QIR;
‘ionq’ – InoQ Json format;
‘cirq_json’ – Cirq Json format;
‘qasm_cirq_compatible’ – OpenQASM 2.0 compatible for Cirq

Timeouts: Generation timeout and Optimization timeout are the two timeouts available to the user. Both are specified in a whole number of seconds.

The generation timeout governs the entire synthesis process while user can set the optimization timeout to restrict the amount of time spend on optimization. In case of no optimization, this timeout will have no effect. Best solution is returned on reaching the optimization timeout. User is informed if the engine hasn’t found a best solution within given time.

User can provide both timeouts on one condition only:

optimization timeout < generation timeout.

1.4 Hardware-Aware Synthesis:

Numerous crucial aspects of quantum computers vary from one another, including basis gates, connection, error rates, etc. The possibility of executing the quantum circuit depends on the device specifications, and logically-comparable programs may need various implementations to maximize the probability of success.

Classiq allows us to describe the hardware we want to utilize to operate our circuit on its platform. The hardware's parameters are taken into consideration by the synthesis engine.

For instance, given the connection of the hardware, the engine might opt for the implementation of a function that requires least swaps.

Specifying a Backend: We can specify the name of the backend and its provider if we wish to synthesize our circuit for a particular backend. Classiq supports the following backend providers: IBM Quantum, Azure Quantum and AWS Braket.

```
{
        "logic_flow": [],
        "preferences": {
                "backend_service_provider": "IBM Quantum",
                "backend_name": "Lima"
        }
}
```

Custom Hardware Settings: We can specify the custom settings of the desirable hardware which is not present in the Classiq platform. These setting include the basis gate and the connectivity map of the hardware. All hardware parameters are optional.

- Basis Gate Set: Single-qubit gates, basic two-qubit gates, extra two-qubit gates and three-qubit gates. Default set consists of all single-qubit gates and basic two-qubit gates.
- Connectivity Map: A collection of qubit ID pair list serves as the connectivity map. Each pair in the list indicates that the pair of qubits can be used for a two-qubit gate, such as cx. Both qubits can act as control if the coupling map is symmetric. The first qubit can only function as control, and the second qubit can only function as target if the coupling map is asymmetric. By default, the engine assumes full connectivity when the connectivity map is not specified.

1.5 Modifying Functions:

Control: It is the quantum analog of classical ‘if’ condition. Superposition principle limits the implementation of classical if-then-else logic on quantum computers.

1.6 Advanced Usage:

Register Slicing: Similar to Python sequences like lists and tuples, the Classiq platform supports referencing to specific qubits or qubit ranges of quantum registers.

The indexing starts at zero. Slicing, also known as referring to multiple consecutive indices, is accomplished by defining the start (inclusive) and stop (exclusive) indices and using a colon separator.

2. Combinatorial Optimization: Classiq’s quantum software engine is a combinatorial optimization platform for user defined optimization problems. It enables users to develop practical optimization problems and create unique quantum circuits to solve them. The user can select to run circuits on a classical simulation or a quantum backend and obtain the optimal result. The created circuit can be analyzed with Classiq's analyzer module to learn more about the synthesis process.

A diverse group of users from different backgrounds can reap the benefits of the Classiq platform.

- Solution strategy is highly customizable according to user’s preferences.
- Default solutions are available that don’t require any quantum expertise.

Problem Formulation:

A new optimization problem is formulated using PYOMO (python based open-source optimization modeling language). PYOMO supports a variety of problem types:

- Integer Linear programming
- Quadratic programing
- Graph theory problems
- SAT problems and much more

2.1 Solve Optimization Problem:

The core capabilities of Classiq:

- Generation of a designated quantum solution
- Execution of the generated algorithm on a quantum backend

PYOMO model (QAOA and optimizer preferences) using following commands:

- generate – generates the quantum circuit
- solve – solves the optimization problem
- get_operator – returns the Ising hamiltonian representing the problem's objective
- get_objective – returns the PYOMO object representing the problem's objective
- get_initial_point – returns the initial parameters for a parametric ansatz
- classical_solution.solve – classically solves the optimization problem

Firstly, the desired optimization problem is formulated using PYOMO model and then the model is sent to the Classiq backend using CombinatorialOptimization package (through its ‘synthesize’ & ‘solve’ commands).

Model Designer command: It exposes the functional-level model of the resulting ansatz.

```
model_designer = mis_problem.get_model_designer()
model_designer = mis_problem.model_designer
```

3. Quantum Algorithms: The hybrid quantum-classical optimization algorithms are implemented through the optimization platform. These algorithms were created to be compatible with the NISQ-era quantum computers with constrained quantum resources.

The algorithms could be split into two categories:

- Variational parameterized quantum circuit also known as an 'ansatz'. An ansatz encodes the possible problem solutions.
- A classical optimization procedure

VQE (Variational Quantum Eigensolver) and QAOA (Quantum Approximate Optimization Algorithm) are two of the most well-known algorithms. A new variant of QAOA (Quantum Alternating Optimization Ansatz) algorithm with same name acronym is introduced. This algorithm restricts the search process of entire Hilbert space to just a subspace. This way the problem constraints are satisfied corresponding to solutions thus reducing the search space by many folds, and ultimately the search performance may enhance. The Classiq platform offers two options for the embedding and analysis of problem constraints.

Original QAOA – In the objective function, penalty terms represent the constraints. Solutions that violate the constraints are discouraged during the optimization process. The algorithm combines every potential solution, and the search space encompasses the entire Hilbert space.

```
qaoa_preferences.qsolver = Qsolve.QAOAPenalty
```

New QAOA – In line with the problem constraints, the search space is constrained explicitly in this algorithm. The constraints are embedded in the initialization and mixer layers.

qaoa_preferences.qsolver = Qsolve.QAOAMixer

4. Solver Customization: The algorithms on the Classiq platform stand out for their flexibility and range of customization possibilities. Both QAOA and new QAOA are determined by a number of important parameters which may be modified using the additional qsolver_preferences and optimizer_preferences inputs to the CombinatorialOptimization class declaration.

The two quantum solvers available to user for implementation are QSolver.QAOAPenalty and QSolver.QAOAMixer in the QAOAPreferences input. The user also specifies the additional parameters of the QAOA ansatz:

1. qaoa_reps – (positive int) Number of layers in QAOA ansatz.
2. penalty_energy - (float) Penalty energy for invalid solutions. The convergent rate is impacted by the value. Small positive values are preferred.
3. initial_state - (List[int]) Custom initial state in QAOA ansatz. By default, it is set to |+> states in QAOAPenalty, and an arbitrary feasible solution in QAOAMixer.

The VQE scheme execution parameters are organized by the optimizer_preferences input. It consists from the following parameters:

1. name - (OptimizerType) Classical optimization algorithms: COBYLA, SPSA, ADAM, L-BFGS-B, NELDER MEAD.
2. num_shots – (positive int) Number of measurements of the ansatz for each assignment of variational parameters.
3. cost_type - (CostType) Summarizing method of the measured bit strings: AVERAGE, MIN, CVAR.

4. alpha_cvar - (positive float) Parameter describing the quantile considered in the CVAR expectation value.
5. max_iteration – (positive int) Maximal number of optimizer iterations.
6. tolerance – (positive float) Final accuracy of the optimization.
7. step_size - (positive float) Step size for numerically calculating the gradient in L_BFGS_B and ADAM optimizers.
8. initial_point - (List of floats) Initial values for the ansatz parameters.
9. skip_compute_variance - (bool) If True, the optimizer will not compute the variance of the ansatz. If no value was provided, appropriate values would be assigned: a random choice for QSolver.QAOAMixer, and a linear schedule for QSolver.QAOAPenalty.

5. User Defined Ansatz: Solving a combinatorial optimization problem with user-defined ansatz and VQE method:

Ansatz Initialization:

```
from classiq import ModelDesigner
from classiq.interface.generator.hardware_efficient_ansatz import
(HardwareEfficientAnsatz,)
model_designer = ModelDesigner()
model_designer.HardwareEfficientAnsatz(HardwareEfficientAnsatz(num_qubits=5,
reps=1))
ansatz = model_designer.synthesize()
```

Using CombinatorialOptimization class all the components are connected and a solve command is sent. QSolver.Custom is the appropriate solve in this case.

```
from classiq.applications.combinatorial_optimization import CombinatorialOptimization
from classiq.interface.combinatorial_optimization.preferences import QAOAPreferences
from classiq.interface.combinatorial_optimization.solver_types import QSolver
```

```
problem = CombinatorialOptimization(
    model=mvc_model,
    qsolver_preferences=QAOAPreferences(qsolver=QSolver.Custom),
    ansatz=ansatz,)
result = problem.solve()
```

Changing the Ansatz:

Another option is to initialize CombinatorialOptimization class with a builtin QSolver and then change it to a custom ansatz.

```
problem = CombinatorialOptimization(
    model=mvc_model,
    qsolver_preferences=QAOAPreferences(qsolver=QSolver.QAOAPenalty),)
problem.ansatz = ansatz
result = problem.solve()
```

Checking Solution Validity:

QAOAMixer is an example for an ansatz which assures that all the resulting solutions are valid under the problem's constraints. The user defined ansatz may have this quality, or not.

CombinatorialOptimization lets the user to declare if the ansatz is "constraints-preserving", and will act accordingly. Namely, for those ansatze (e.g. QAOAMixer), we will check if all the solutions are valid. This attribute is defined using should_check_valid_solutions field.

The default value of should_check_valid_solutions field is set to True only for QSolve.QAOAMixer and for a quantum backend which is an exact simulator. If the backend is noisy we cannot completely assure the solutions validity.

So, for example, if we first use QAOAMixer and then switch to a user-defined ansatz which doesn't preserve the constraints, we have to explicitly switch off the should_check_valid_solutions field.

```
problem = CombinatorialOptimization(
   model=mvc_model,
   qsolver_preferences=QAOAPreferences(qsolver=QSolver.QAOAMixer),
)
problem.should_check_valid_solutions = False
problem.ansatz = ansatz
result = problem.solve()
```

E. QUANTASTICA

- Prepare Input Data
- Encode Data into State Vectors
- Find Unitary Transformation
- Prepare for Scaling
- Scale-up

3. NEW INITIATIVES FOR USER CENTRIC AUTOMATION OF EVERYTHING

A. Problems with Current Automation Systems:

(a) Current automaton system is expert centric i.e. it requires an expert to give instructions and control events in the process than allowing any illiterate user also to give real life inputs in the form of touch, image, audio, video, text etc. Some initiatives are being taken to let any user give inputs but their pipelines doesn’t generate the required quantum algorithm.

(b) It is industry centric because only those who have huge resources can dream to create complex softwares and hardwares requiring huge investments. This means that people have no choice to compete with the tech giants.

(c) It drifts all the resources from the bottom to the top due to master-slave philosophy, custodian centric design of institutions, capital intensive development model, right to private property law, asymmetric information flow etc.

B. Reinventing the Conceptual Foundations for User Centric Automation of Everything:

(a) If we want to benefit everybody from the automation of everything then we need to make it accessible to everybody by automating entire ML pipeline such that any user may give her/his real life inputs in the form of touch, image, audio, video, text etc. Besides this, technology has to be developed for automated data collection e.g. automated data collection from human cell, inside earth and sea, and outer space etc.

(b) We need to ground user centric automaton in a new development model valuing online labor, creativity and computation so that anyone may earn 100% profits from her/his choice online activity.

(c) User centric automation of everything will require to make everything computable that means we need better and better models of computation.

C. Data Driven Automation of Entire Machine Learning Pipeline:

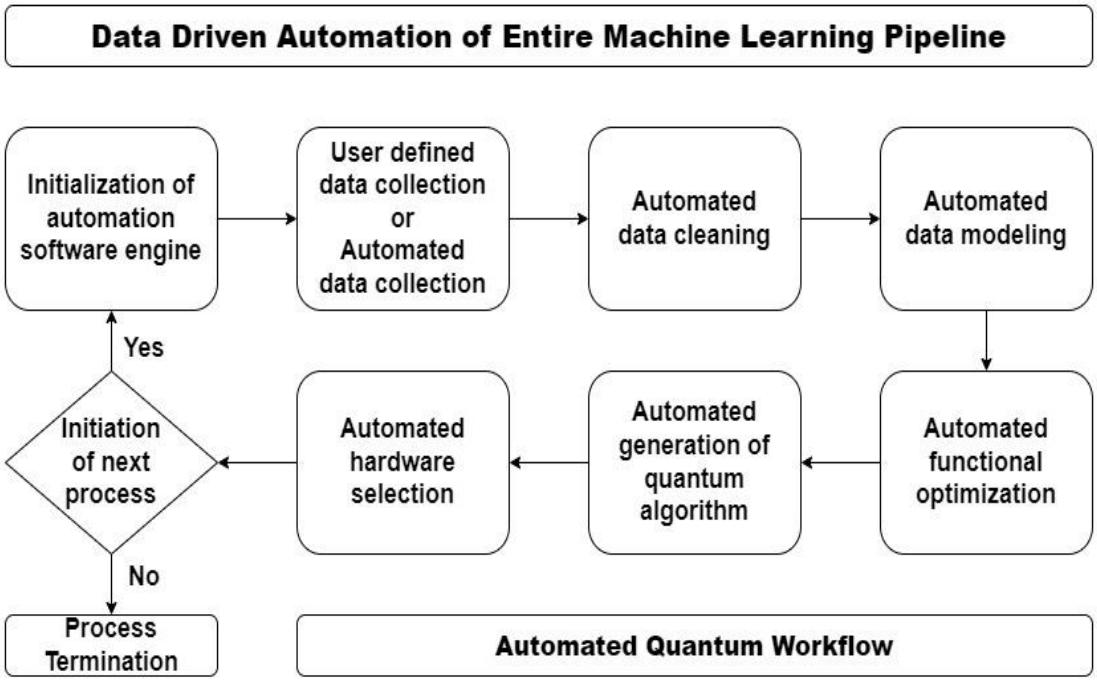

Fig 3.1: Data Driven Automation of Entire ML Pipeline

- **Automated Data Collection:**

  Quantum Sensors work on the principle of quantum entanglement, quantum interference and quantum state squeezing with optimized precision and beat the current limits of sensor technology.

  Quantum sensing is a technological field which deals with the design and engineering of quantum sources and quantum measurements that can beat any classical strategy in various applications.

  Quantum Sensors are often built on continuously variable systems (quantum systems characterized by continuous degrees of freedom such as position & momentum). Basic working mechanism of quantum sensor typically relies on optical states of light (involving mechanical properties such as squeezing or two-mode entanglement).

- **Automated Data Cleaning:**

Using machine learning, the data is modified or removed if it is incorrectly formatted, incorrect, incomplete, duplicated or flawed. On an estimate, manual data cleaning consumes almost 90% of the time spent in the machine learning life cycle. This is a serious cause of concern as all the focus is fixated on making the data conducive rather than on maximizing the potential of other parts of machine learning i.e. algorithm selection / generation, hyper parameter optimization / tuning, algorithm evaluation, data specific hardware selection etc.

Now we need to answer an important question: Which steps of data cleaning can be actually standardized and automated? The answer to this question is: the steps which are performed frequently (time and again) are most likely to be automated or standardized. Next, we want to ensure that our pipeline is generic and can adapt to various types of datasets.

1. CSV / JSON – There are five types of building blocks for this type of data. These are processed as follows:

Missing Values (Block 1): There are two ways to deal with missing values, we either delete those observations that contains missing values or we can simply use an imputation technique. An imputation technique replaces missing data by certain values, like mean, median, mode, or by a value similar to other values of the sample. Classification or Regression models are quite useful in prediction of missing values in our data.

To handle the missing values, we primarily focus on:

- the data type (numerical or categorical)
- count of missing values relative to the count of total samples

We will adopt the strategy to give more preference to imputation than deletion. We will be focusing on following strategies: prediction with linear and logistic regression and imputation & deletion with mean, median and mode as well as K-NN.

The class structure of missing values is as follows:

```
class MissingValues:
    def handle(df, missing_num='auto', missing_categ='auto', _n_neighbors=3):
        ...
    def _impute(df, imputer, type):
        ...
    def _lin_regression_impute(df, model):
        ...
    def _log_regression_impute(df, model):
        ...
    def _delete(df, type):
        ...
```

Table 3.1: Missing Values Class Structure

The handle function will handle numerical and categorical missing values in the following way: some imputation techniques are applicable only for numerical data while some are applicable only for categorical data. Handle function firstly checks for the handling method which is by default set to ‘auto’ indicating:

If the missing data is of numerical type then the data is either deleted or imputed through

- linear regression or
- knn or
- mean, median or mode

If the missing data is of categorical type then the data is either deleted or imputed through

- logistic regression or
- knn or
- mode

It is to be noted that the categorical data will be first label encoded to integers to be able to predict the missing values and finally the labels must be mapped back to their original values.

The impute function will impute the missing values in the data through knn, mean, median and mode while the linear and logistic regression function will impute the missing values through prediction. Lastly, the delete function will delete the missing values in the data.

Outliers (Block 2): Outliers are the extreme values in a dataset. Firstly, we need to finalize a criteria to label a value to be an outlier. We will consider a value or a data point to be an outlier if it does not fall inside the following range:

$$Q1 - 1.5 * IQR; Q3 + 1.5 * IQR$$

where Q1 and Q3 are the 1^{st} and the 3^{rd} quartiles and IQR is the interquartile range.

Much like the missing values problem, there are two ways to deal with outliers namely, winsorization and deletion. Winsorization is a statistical technique used to limit extreme values in the data to limit the effect of the outliers. In addition, outliers are replaced with a specified percentile of the data. Outliers will be replaced using above defined range through winsorization:

- upper range value will replace values > upper bound and
- lower range value will replace values < lower bound

The class structure of outliers is as follows:

```
class Outliers:
    def handle(df, outliers='winz'):
        if outliers:
            if outliers == 'winz':
                df = Outliers._winsorization(self, df)
            elif ourliers == 'delete':
                df = Outliers._delete(self, df)
        return df
    def _winsorization(df):
        ...
    def _delete(df):
        ...
    def _compute_bounds(df, feature):
        ...
```

Table 3.2: Outliers Class Structure

The handle function will handle the outliers in the data in two ways: winsorization or deletion. The winsoriation function will compute the upper bound and lower bound and check for the two conditions mentioned above. Whichever conditions matches, that value will replace the respective outlier with an appropriate value. The delete function will delete the outliers in the data. Lastly, the compute bounds function will compue the lower and upper bounds for finding the outliers in the data.

Categorical Encoding (Block 3): There are no problems while handling numerical type data but categorical data first needs to be changed into numerical data to perform

any computations. There are two common techniques available to change categorical data to numerical data namely, label encoding or one-hot encoding.

In label encoding, a categorical value is replaced with a numerical value ranging between 0 and total classes – 1. Label encoding assigns a unique integer to each value. It has one disadvantage that the numerical values can be misinterpreted by algorithms as having some kind of order of hierarchy in them. A contemporary approach of one-hot encoding is present to address the above mentioned issue.

In one-hot encoding, a categorical value is replaced with a binary (0 or 1) value. Although, this approach solves the problem of order / hierarchy but it has its own disadvantage of adding more columns to the data set (if many unique values are present).

Categorical encoding will be done keeping in mind three rules, with the default strategy set to 'auto', that are:

- one-hot encoded if the feature contains < 10 unique values
- label-encoded if the feature contains < 20 unique values
- not encoded if the feature contains > 20 unique values

The class structure of categorical encoding is as follows:

```
class EncodeCateg:
    def handle(df, encode_categ=['auto']):
        if encode_categ[0]:

            cols_categ = set(df.columns) ^
set(df.select_dtypes(include=np.number).columns)
```

```
        if len(encode_categ) == 1:
            target_cols = cols_categ
        else:
            target_cols = encode_categ[1]
        for feature in target_cols:
            if feature in cols_categ:
                feature = feature
            else:
                feature = df.columns[feature]
            try:

                pd.to_datetime(df[feature])
            except:
                try:
                    if encode_categ[0] == 'auto':

                        if df[feature].nunique() <=10:
                            df = EncodeCateg._to_onehot(df, feature)

                        elif df[feature].nunique() <=20:
                            df = EncodeCateg._to_label(df, feature)

                    elif encode_categ[0] == 'onehot':
                        df = EncodeCateg._to_onehot(df, feature)
                    elif encode_categ[0] == 'label':
                        df = EncodeCateg._to_label(df, feature)
                except:
                    pass
    return df
def _to_onehot(df, feature, limit=10):
```

```
    ...
def _to_label(df, feature):
    ...
```

Table 3.3: Categorical Encoding Class Structure

Extraction of DateTime features (Block 4): Extraction of features from the dataset that have datetime values becomes easier to handle while processing or visualizing them later. Timestamps or dates are some of these datetime values.

This can be done automatically by searching through the features with the help of pipeline and checking whether conversion into the datetime type is possible for any one of them.

The class structure of datetime is as follows:

```
def convert_datetime(df, extract_datetime='s'):
    cols = set(df.columns) ^ set(df.select_dtypes(include=np.number).columns)
    for feature in cols:
        try:
            df[feature] = pd.to_datetime(df[feature], infer_datetime_format=True)
            df['Day'] = pd.to_datetime(df[feature]).dt.day
            if extract_datetime in ['M','Y','h','m','s']:
                df['Month'] = pd.to_datetime(df[feature]).dt.month
                if extract_datetime in ['Y','h','m','s']:
                    df['Year'] = pd.to_datetime(df[feature]).dt.year
                    if extract_datetime in ['h','m','s']:
                        df['Hour'] = pd.to_datetime(df[feature]).dt.hour
                        if extract_datetime in ['m','s']:
                            df['Minute'] = pd.to_datetime(df[feature]).dt.minute
                            if extract_datetime in ['s']:
```

```
                    df['Sec'] = pd.to_datetime(df[feature]).dt.second
        try:
            if (df['Hour'] == 0).all() and (df['Minute'] == 0).all() and (df['Sec'] == 0).all():
                df.drop('Hour', inplace = True, axis =1 )
                df.drop('Minute', inplace = True, axis =1 )
                df.drop('Sec', inplace = True, axis =1 )
            elif (df['Day'] == 0).all() and (df['Month'] == 0).all() and (df['Year'] == 0).all():
                df.drop('Day', inplace = True, axis =1 )
                df.drop('Month', inplace = True, axis =1 )
                df.drop('Year', inplace = True, axis =1 )
        except:
            pass
    except:
        pass
return df
```

Table 3.4: DateTime Class Structure

By default, the granularity is set to ‘s’ for seconds. The granularity can be customized too to extract the datetime features. After the extraction, the date and time entries are checked for correctness by the function. If the extracted columns namely ‘Day’, ‘Month’ and ‘Year’ contain 0’s then all three of them will be deleted. Same goes for ‘Hour’, ‘Minute’ and ‘Sec’.

Dataframe Polishing (Block 5): Although, the dataset is processed, some more modifications are still needed to make the dataframe ‘look good’. Firstly, some features that were initially of type integer may have been transformed to floats through imputation techniques or other processing procedures, so before outputting the final dataframe these values will be converted back to integers.

Secondly, all float values will be rounded to same number of decimals. This will ensure that float decimals are free from unnecessary trailing 0's and the values are not rounded greater than the original values.

2. Text Data – In order for machines to understand human language, raw text must be cleaned before it can be used for natural language processing (NLP).
Clean text is simply human language that has been organized in a way that computer models can interpret. Simple Python code that removes stop words, Unicode words, and breaks down complex words to their root form can be used to clean up text.

Normalizing Text: The method of normalizing text involves standardizing text so that, through NLP, human input is better understandable to computer models. This will result in effectively performing sentiment analysis. Text normalization through NLTK library involves standardizing capitalization so that capitalized words are not grouped together by machine models. This way lowercase and uppercase words will be segregated.

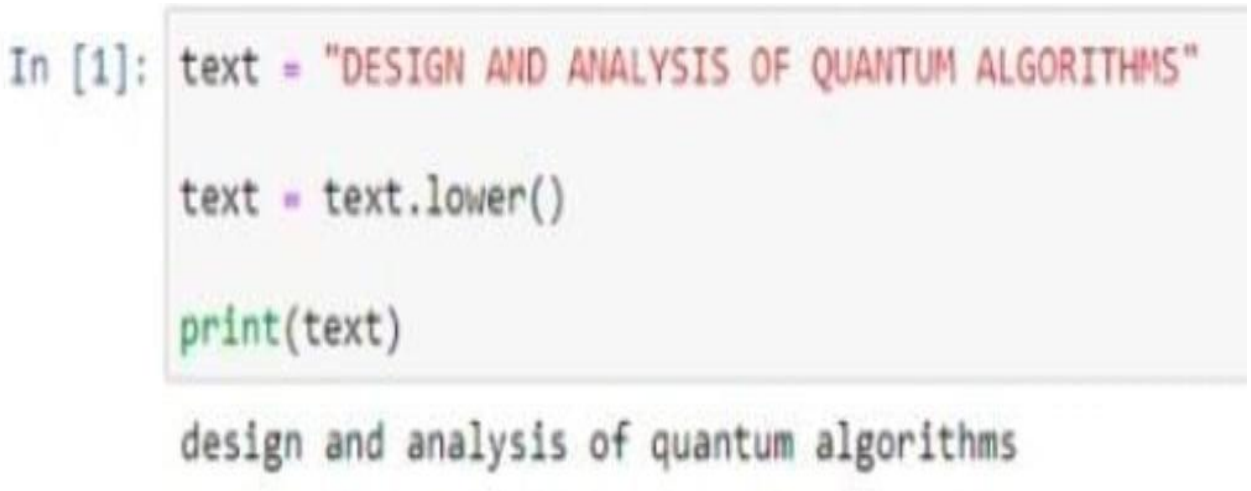

Fig 3.2: Text Normalization

Removing Unicode Characters: Punctuation, Emoji, URLs, and @'s are unique signatures that either end up being translated unhelpfully into unicode (the smiley face

becomes \u200c or something similar), or are unique (in the case of @'s and hyperlinks), which causes AI models to become confused. Punctuation adds noise and hinders NLP comprehension because it affects the tone of the sentence as a whole rather than just the word it is related to.

```
In [2]: import re

        text = "Design and Analysis of Quantum Algorithms! https://qiskit.org/textbook/ch-algorithms/"

        text = re.sub(r"(@\[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)|^rt|http.+?", "", text)

        print(text)

Design and Analysis of Quantum Algorithms
```

Fig 3.3: Removing Unicode Characters

Removing Stopwords: Stop words are frequent terms within phrases that do not contribute value and can be removed when cleaning for NLP prior to analysis. Stop words are common in English and in many other widely used languages.

```
In [3]: import nltk.corpus
        nltk.download('stopwords')
        from nltk.corpus import stopwords

        stop = stopwords.words('english')
        text = "Stop words are recurrent within sentences that don't provide any value"
        text = " ".join(\
                        [word for word in text.split() if word not in (stop)])

        print(text)

Stop words recurrent within sentences don't provide value
```

Fig 3.4: Removing Stopwords

Stemming and Lemmatization: It entails dissecting words to reveal their roots and related root meanings. We can more accurately gauge purpose if we do this. It is crucial to choose the right technique for the analysis we want to undertake because, despite the similarities between the two techniques, they yield very different outcomes.
Stemming arranges words according to their root stem. This enables us to see that the words "playing," "plays," and "played" are all derived from the same verb (play), and as a result, refer to issues that are comparable.

Lemmatization groups words according to the definition of their roots and enables us to distinguish between the present, past, and indefinite.

In contrast to all instances of "played," which are gathered together as past tense, and all instances of "playing," which are grouped together as the indefinite (meaning continuing/continuous), "plays" and "play" are therefore grouped into the present "play."

Therefore, stemming would be used if we wanted to locate every instance of a product having any kind of "play"-related response so we could assess all replies, positive or negative.

However, if we want to further dissect this to the type of play, i.e. whether it was a continuous issue, past issue, or present issue, and want to tackle each of these three occurrences with a different type of analysis, then we will employ lemmatizing.

In [4]:
```
import nltk
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
words = \
["play", "played", "plays", "playing"]
stemmer = PorterStemmer()
for word in words:

    print(word + " = " + stemmer.stem(word))
```

```
play = play
played = play
plays = play
playing = play
```

In [5]:
```
import nltk
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
words = \
["play", "played", "plays", "playing"]
lemmatizer = WordNetLemmatizer()
for word in words:

    print(word + " = " + lemmatizer.lemmatize(word))
```

```
play = play
played = played
plays = play
playing = playing
```

Fig 3.5: Stemming and Lemmatization

- **Automated Data Modeling**

Pyomo stands for Python Optimization Modeling Objects. It is a modeling tool based on python programming language. It is used for model optimization. It comes under the umbrella of algebraic modeling languages (AMLs).

An abstract (or symbolic) mathematical model is defined using symbols representing data values. Abstract models rely on unspecified parameter values. A model instance can be specified using data values.

A concrete mathematical model is defined directly with data values that were assigned when the model was defined.

Variables: A numerical value determined during optimization. The search space for optimization is defined by the variables.

Pyomo variables are managed with the Var class, which can denote a single, independent value, or an array of values. Variables can have initial values, and the value of a variable can be retrieved and set.

Declarartion of a single variable: model.x = Var()

Declaration of multi-dimensional variable arrays:

```
B = [1.5, 2.5, 3.5]
model.u = Var(B)
model.C = Set()
model.t = Var(B, model.C)
```

Using either the domain option or within option variable domain is specified:

```
model.A = Set(initialize=[1,2,3])
model.y = Var(within=model.A)
model.r = Var(domain=Reals)
model.w = Var(within=Boolean)
```

In case, no domain is specified then the default option Any is used. Virtual sets are most common domains that are used.

The bounds option is used to specify the variable bounds. The bounds option specifies a 2-tuple (having lower and upper values). “Alternatively, it can specify a function that returns a 2-tuple for each variable index”. When a bound is not enforced then it is specified using None option.

The initialize option sets the initial values of variables.

```
model.za = Var(initialize=9, within=NonNegativeReals)
model.zb = Var(model.A, initialize={1:1, 2:4, 3:9})
model.zc = Var(model.A, initialize=2)
```

Additionally, this option can use a function that accepts the variable indices and model and returns the value of that variable element:

```
def g(model, i):
return 3*i
model.m = Var(model.A, initialize=g)
```

Using the reset method, variable's value is set to the initial value:

```
model.za = 8.5
print model.za.value # 8.5
model.za.reset()
print model.za.value # 9
```

Using the equality operator, variable values can be set:

```
model.za = 8.5
model.zb[2] = 7
```

For passing values to a solver, an alternative of initialize option is the equality operator for passing the initial values.

Helper functions and utility methods facilitate the use of variable objects. The float function and the value function are used to coerce a Var object into a floating point value and numerical value respectively.

```
print float(model.za) # 8.5
print float(model.zb[2]) # 7.0
print value(model.za) # 8.5
print value(model.zb[2]) # 7
```

In a variable array, the number of variables are determined through len function

```
print len(model.za) # 1
print len(model.zb) # 3
```

Interacting directly with Var object is useful in many ways. These objects have a variety of useful attributes that can be queried and set:

print model.zb[2].value # 7
print model.zb[2].initial # 4
print model.za.lb # 0
print model.za.ub # None
print model.zb[2].fixed # False

Variable's current value – value attribute
Variable's initialized value – initial attribute
Variable's upper bound value – ub attribute
Variable's lower bound value – lb attribute
Fixed variable value – fixed attribute

These values may be preprocessed by Pyomo before calling the optimizer. These attributes are Var object for singleton variables.

Pyomo has a common design pattern where a clear difference between component object and component value objects is seen. These two classes are in combined form for singleton components while they are different for indexed components.

Objectives: An objective is a function. This function involves data and variables which is either maximized or minimized through a solver. The solver looks for the variable's values that provide the best possible value an objective function can have. Although, constraints may limit the possible values for the variables, the objective function dictates how the variable's values will be assessed.

Declaration of a single objective: model.a = Objective()

Declaration of multiple objectives:

```
model.b = Objective()
model.c = Objective([1,2,3])
```

Without a functional expression declaration, above examples are incomplete.

The characteristics of the objective function, such as the functional expression and a declaration of whether it should be minimized or maximized, are defined by an Objective object. Assume that the variables model.x[1] and model.x[2] have been declared. The objective for the model is defined in the following declaration with a function that is the sum of the first variable and two times of the second variable:

```
model.d = Objective(expr=model.x[1] + 2*model.x[2])
```

By default, the objective is minimized but with the use of sense option an objective can be maximized too.

```
model.e = Objective(expr=model.x[1], sense=maximize)
```

Declaration with Rule Functions: Using the expr option, it can be specified that which function needs to be optimized. Writing a rule function is an alternative for initializing an Objective object. This rule function defines the objective function by creating an expression. The name of the rule function is specified through the rule option. Two equivalent declarations of Objective object are as follows:

```
model.f = Objective(expr=model.x[1] + 2*model.x[2])
def TheObjective(model):
return model.x[1] + 2*model.x[2]
model.g = Objective(rule=TheObjective)
```

```
def gg_rule(model):
return model.x[1] + 2*model.x[2]
model.gg = Objective()
```

Pyomo iterates over all set members when the Objective object is declared with a set as an argument, passing each member of the set to the function specified as the argument to the rule keyword.

```
def h_rule(model, i):
return i*model.x[1] + i*i*model.x[2]
model.h = Objective([1, 2, 3, 4])
```

In an Objective declaration, when multiple sets are specified then Pyomo iterates over the cross product of all input sets giving the rule function an index for each set. The rule function receives the model that is currently being built as its first argument.
The expr option cannot be used to initialize the h objective defined in this last example. The rule option is needed to define functions for each of the objectives. In some situations the rule option is preferable even for single objective declarations.

Specifically, a rule function provides control over how the objective is formed or used to build up the expression. For example, the following rule function illustrates how an expression is constructed incrementally:

```
def m_rule(model):
expr = model.x[1]
expr += 2*model.x[2]
return expr
model.m = Objective()
```

Similarly, the following rule function illustrates the use of scripting to control the creation of the expression:

```
p = 0.6
def n_rule(model):
if p > 0.5:
return model.x[1] + 2*model.x[2]
else:
return model.x[1] + 3*model.x[2]
return expr
model.n = Objective()
```

In the interest of completeness, we note that this objective could also be done in a concrete model without a rule function in the following way:

```
p = 0.6
if p > 0.5:
model.p = Objective(expr=model.x[1] + 2*model.x[2])
else:
model.p = Objective(expr=model.x[1] + 3*model.x[2])
```

Constraints: Expressions that place a limit on the values of variables are defined by constraints. Variable values can also be limited using the bounds option, however constraints broaden this concept by enabling expressions that limit a number of interacting variables at once.

Constraints expressions are declared just like the objective function expressions. However, there is one stark difference between constrains and objectives, that is, for specifying the equalities or inequalities some auxiliary information needs to be augmented to the expressions.

Another difference between constraints and objectives is in the terms of indexing. Constraints are frequently indexed, enabling access to variables and indexed parameters while creating the constraint expression. Although objectives can be indexed, this feature isn't often used.

Declaration of a non-indexed Constraint object:

model.Diff= Constraint(expr=model.x[2]-model.x[1] <= 7.5)

The expr keyword specifies an expression, but a rule function can also be used to generate it, when an objective function is declared. The Diff constraint can also be created as follows:

```
def Diff_rule(model):
return model.x[2] - model.x[1] <= 7.5
model.Diff = Constraint()
```

Declarations with Expression Tuples: Constraints

A tuple (l, f, u) can be used to specify constraint expressions apart from logical expressions. A tuple equivalent to $l \leq f \leq u$ or a tuple (l, f) equivalent to l == f.

With very few exceptions, l and u must contain just parameters or constants, while an expression for f can comprise constants, parameters, and variables. Given a tuple, the keyword None can specify either l or u for indicating a one-sided constraint. Both l and u can’t be specified None value simultaneously. This would indicate a constraint expression with an unbounded scope, which is not really a constraint.

The most computationally effective technique to specify constraints in Pyomo is to use tuples. Constraints expressed as equalities and inequalities need to be re-structured to

adhere to the tuple representation because Constraint objects use it internally. Non-trivial computation is required for this re-structuring.

Tuple value interpreted as inequalities:

```
def CapacityIneq_rule(model, i):
return (0.25, (a[i] * model.y[i])/b[i], 1.0)
model.CapacityIneq = Constraint(N)
```

This constraint captures the mathematical concept – $0.25 \leq (a_i x_i) / b_i \leq 1, \forall i \in \{1, 2, 3\}$ is equivalent to a times x should be between one-fourth b and one unit of b.

```
def CapacityEq_rule(model, i):
return (0, a[i] * model.y[i] - b[i])
model.CapacityEq = Constraint(N)
```

Specifies an equality constraint using the tuple syntax, encoding the mathematical concept:

$0 \leq a_i x_i - b_i \leq 0 \ \forall i \in \{1,2,3$ (3.3) or, after rearranging terms, that a times x must equal b.

When a constraint is declared, each member of the cross product of the index sets utilised is called by the rule function. In some optimization models, there may be the case that constraints are not defined for all indices. It may be the case that the physical realization is not possible for particular indices. If there are no constraints associated with a certain index, then the rule function indicates this by returning the constant value. This constant value is returned through Constraint.Skip (or Constraint.NoConstraint) option.

Model Data: The two components of pyomo model namely set and param are used to construct constraints and objectives (for defining the data). By default, abstract data declarations are defined through set and param components. Declarations for set A and parameter p (with no data):

model.A = Set (within = Reals)
model.p = Param (model.A, within = Integers)

Set component is not abstract if there is no specification of initialize option	Param component is not abstract if there is no specification of initialize option as well as default option
Set or Param component is abstract if indexing is done through an abstract Set component	

Table 3.5: Set and Parameter

Data command file: A sequence of commands directly specifying set and parameter data or specifying external sources to obtain such data. Commands for data declaration:

- set command – set data
- param command – parameter data table. It can also include the declaration of the set data used to index parameter data
- table command – 2-dimensinal table of parameter data
- import command – importing set and parameter data through external data sources. It includes ASCII table files, CSV files, ranges in spreadsheets and database tables

Some more commands used in data command files:

- include command – specifying a data command file that needs to be processed immediately.

- data and end commands – they perform no action. Both the commands provide compatibility with AMPL (A Mathematical Programming Language) scripts to define data commands
- namespace keyword – data commands are organized into named groups and during model construction these groups can be enabled or disabled.

Pyomo data commands:

- are terminated with a semicolon
- does not depend on whitespace
- can be broken over multiple lines
- newlines and tab characters are ignored
- formatted with whitespace (involving few restrictions)

The set command

Simple sets: set command specifies explicitly the members of either a single set or an array of sets (an indexed set). A single set is specified with a list of data values that are included in this set. Syntax: set <setname> := [<value >] ... ;

Data values in a set:

- Numeric values – strings that are evaluated by Python as numeric value (integer, float, boolean or scientific notation
- Simple strings – sequences of alpha-numeric characters
- Quoted strings – strings inside a pair of single or double quotes. It can include quotes within the quoted string.

A set declaration has no restrictions regarding values in a set. A set can be empty and it can contain different combination of numeric and non-numeric string values. When

constructing a Pyomo model, set data validation is performed. Set data is not validated while parsing a data command file. Few examples of valid set commands:

empty set	set A := ;
numbers set	set A := 1 2 3;
strings set	set B := north south east west;
mixed types set	set C := 0 -1.0e+10 'foo bar' infinity "100";

Table 3.6: Types of Set

- When the set data specification is parsed, numerical values automatically convert to integer or floating point values.
- A string value containing a numeric value can be defined through a quoted string. Although, a string strictly specifying a numeric value will get converted to a numeric type (by Python).

Tuple Data sets:

Specifying tuple data – model.A = Set (dimen=3)

Specifying that A is the containing the tuples – set A := (10, 9, 8) (7, 6, 5);

Alternative way of listing the order in which tuple is represented – set A := 10 9 8 7 6;

2-tuple data can be specified in a matrix denoting set membership. An example of 2-tuples in A with valid tuples (+) and invalid tuples (-).

```
set A : A1 A2 A3 A4 :=
1 + - - +
2 + - + -
3 - + - -  ;
```

Following five tuples are declared using the above data command:

(‘A1’, 1), (‘A1’, 2), (‘A2’, 3), (‘A3’, 2), (‘A4’, 1)

Tuple template represents a slice of tuple data. Following is the declaration of group of tuples (defined by a template) and data to complete the template:

```
model.A = Set (dimen=4)
set A :=
    (1, 2, *, 4)  A B
    (*, 2, *, 4)  A B C D;
```

In place of a value, the tuple template has a tuple which contains one or more * symbols. These indices represent the tuple value which replaces the values from the list of values that follows the tuple template.

```
(1, 2, ‘A’, 4)
(1, 2, ‘B’, 4)
(‘A’, 2, ‘B’, 4)
(‘C’, 2, ‘D’, 4)
```

Set Arrays: set command can be used to declare data for a set array. Syntax:

set <set−name>[<index >] := [<value >] ... ;

As an arbitrary set can index set arrays, so index value can be a numeric value, non-numeric string value or a comma separated list of string values. Indexing a set B using the values declared for set A:

```
model.A = Set()
model.B = Set(model.A)
set A := 1 aaa ‘a b’;
set B[1] := 0 1 2;
set B[aaa] := aa bb cc;
set B[’a b’] := ‘aa bb cc’;
```

The param command

There are two types of parameter data: one-dimensional and multi-dimensional. Declaration of simple or non-indexed parameters is done in following way:

```
param A := 1.4;
param B := 1;
param C := abc;
param D := true;
param E := 1.0e+04;
```

Parameters can be defined with numeric and string data. A string evaluated by Python as a numeric value (integer, floating point, scientific notation, and Boolean) defines the numeric data. Strings like TRUE, true, True, FALSE, false and False specify the Boolean values. There is no specification of an empty parameter as the parameters can’t be defined without data.

One-dimensional parameter data: Indexed over a single set. List of index-value pairs specifying values for parameter B which is indexed by the set A:

```
model.A = Set()
model.B = Param(model.A)
set A := a c e;
param B := a 10 c 30 e 50;
```

As whitespace is ignored, so reorganizing the same data in a tabular format will look like:

```
set A := a c e;
param B :=
a 10
c 30
e 50;
```

Defining multiple parameters using a single param data command. Let, the set A indexes three one-dimensional parameters B, C and D.

```
model.A = Set()
model.B = Param(model.A)
model.C = Param(model.A)
model.D = Param(model.A)
```

The values can also be specified using a single param data command followed by index and parameter values:

```
set A := a c e;
param : B C D :=
a 10 -1 1.1
c 30 -3 3.3
e 50 -5 5.5;
```

These values are interpreted as a list of sublists. Each sublist consists of an index followed by the corresponding numeric value.

A data command file is totally valid even if the parameter values are not defined for al indices. For ex:

```
set A := a c e g;
param : B C D :=
a 10 -1 1.1
c 30 -3 3.3
e 50 -5 5.5;
```

The values for the index g is not specified. This way the index g becomes invalid for the model instance using this data. The '.' character specifies the complex patterns of missing data for indicating a missing value. The following example shows the representation of parameters with common index set and how each parameter can be indexed differently:

```
set A := a c e;
param : B C D :=
a . -1 1.1
c 30 . 3.3
e 50 -5 .;
```

The data for set A is specified through data file in two ways:

- When A is defined
- Implicitly when parameter are defined

Alternatively, using param command a user can specify the definition of an index set with associated parameters:

```
param : A : B C D :=
a 10 -1 1.1
c 30 -3 3.3
e 50 -5 5.5;
```

Default values can only specify the values for a single parameter through param command. The default keyword can specify the default values for missing data in the following way:

```
set A := a c e;
param B default 0.0 :=
c 30
e 50;
```

Multi-dimensional parameter data: Indexed over multiple sets or multi-dimensional sets. Specifying values for parameter B which is indexed by the set A having dimension two:

```
model.A = Set(dimen=2)
model.B = Param(model.A)
set A := a 1 c 2 e 3;
param B :=
a 1 10
c 2 30
e 3 50;
```

Much like the one-dimensional parameter data, the default values and missing values in multi-dimensional data are handled on the same lines:

```
set A := a 1 c 2 e 3;
param B default 0 :=
a 1 10
c 2 .
e 3 50;
```

Defining multiple parameters using a single param data command. Let, the set A (having dimension 2) indexes three one-dimensional parameters B, C and D.

```
model.A = Set(dimen=2)
model.B = Param(model.A)
model.C = Param(model.A)
model.D = Param(model.A)
```

The values can also be specified using a single param data command followed by index and parameter values:

```
set A := a 1 c 2 e 3;
param : B C D :=
a 1 10 -1 1.1
c 2 30 -3 3.3
e 3 50 -5 5.5;
```

Defining a matrix of parameter values, where set A (dim. 2) indexes parameter B:

```
model.A = Set(dimen=2)
model.B = Param(model.A)
```

```
set A := 1 a 1 c 1 e 2 a 2 c 2 e 3 a 3 c 3 e;
param B : a c e :=
1 1 2 3
2 4 5 6
3 7 8 9;
```

A transpose matrix of parameter values can be specified as follows:

```
set A := 1 a 1 c 1 e 2 a 2 c 2 e 3 a 3 c 3 e;
param B (tr) : 1 2 3 :=
a 1 4 7
c 2 5 8
e 3 6 9;
```

This functionality facilitates the presentation of parameter data in a natural format. In particular, the transpose syntax may allow the specification of tables for which the rows comfortably fit within a single line. However, a matrix may be divided column-wise into shorter rows since the line breaks are not significant in Pyomo's data commands.

It is necessary to specify the parameter data values for parameters having three or more indices. This specification should be in terms of series of slices. A template (followed by index and parameter values) defines each slice. Set A (dimension 4) indexes parameter B and defines a matrix with multiple templates:

```
set A := (a,1,a,1) (a,2,a,2) (b,1,b,1) (b,2,b,2);
param B :=
[*,1,*,1] a a 10 b b 20
[*,2,*,2] a a 30 b b 40
```

The parameter B consists of the following four values: B[a, 1, a, 1] = 10, B[b, 1, b, 1] = 20, B[a, 2, a, 2] = 30, and B[b, 2, b, 2] = 40.

The **table** command

This command specifies explicitly a 2-dimensional array of parameter data. This command has some advantages over the param command in terms of flexibility and complete data declaration. Illustration of a table command for a single parameter data declaration is as follows:

table M(A) :
A B M N :=
A1 B1 4.3 5.3
A2 B2 4.4 5.4
A3 B3 4.5 5.5;

The **import** command

This command allows to import data from various external tabular data sources. In a Pyomo model, set and parameter data are represented through a relational table. Numeric string values matrix, simple strings and quoted strings are all part of a relational table. All rows and columns have the same lengths. Column data labels is represented by the first row.

External data sources that can be loaded through load command are:

- Tab file: text file format. Separate columns
- Csv file: text file format.
- Xml file: extensible markup language. They can represent tabular data.
- Excel file: spreadsheet data format. Used by Microsoft Excel application.

- Database: relational database.

The load command makes use of a data manager to extract data from a specified data source.

In this way, the import command provides a generic mechanism that enables Pyomo models to interact with standard data repositories that are maintained in an application-specific manner.

A basic illustration of the load command (for an indexed parameter) is specifying data. Set A indexes the values of parameter B in following way:

A B
A1 3.3
A2 3.4
A3 3.5
Parameter data loading: import B.tab : [A] B;

Index set A and parameter data loading: import B.tab : A=[A] B;

The difference is the specification of the index set, A=[A], which indicates that set A is initialized with the index loaded from the ASCII table file.

Loading set data from a ASCII table file. This file contains a single data column:

A
A1
A2
A3

Format option specifies that the relational data is depicted as a set:

import A.tab format=set : A;

This allows for specifying set data that contains tuples. There are two parts of the syntax of the load command.

1st part – Starts with a filename, URL of the database or Data Source Name (DSN) and ends with the colon.

2nd part – Specification of column names for indices and data.

Interpreting relational table as a set: import ABCD.tab format=set : Z;

Data is loaded into the model where Z is a set. If Z doesn't exist then while loading data from the table an error will arise.

Interpreting relational table as a parameter:

3-tuple indexing import ABCD.tab : [A,B,C] D;

Data is loaded into the model where D is a parameter. The indices specified in the table are contained in the index set for D.

Index set specification is as follows:

import ABCD.tab : Z=[A,B,C] D;

Loading data into another parameter: import ABCD.tab : Z=[A,B,C] Y=D;

Index set loaded into the Z set and data in the column D loaded into the Y parameter.

Specifying data mappings – from columns to index sets and parameters. How data items are loaded with the help of three columns B, C and D in a model with set Z and two parameters (Y and W) is as follows:

```
model.Z = Set()
model.Y = Param(model.Z)
model.W = Param(model.Z)
import ABCD.tab : Z=[B] Y=D W=C;
```

Now moving on to the using option, if it is omitted then the data manager is inferred from the filename suffix. Sometimes the format of the data is not reflected by the filename suffix.

```
A,B,C,D
A1,B1,1,10
A2,B2,2,20
A3,B3,3,30
```

Importing parameter D and set Z from this file by specifying the using option:

```
import ABCD.txt using=csv : Z=[A,B,C] D;
```

Some valid options in context of the using option “is” the data manager for:

- CSV files (csv keyword)
- ASCII table files (tab keyword) and
- Excel spreadsheet files (xls keyword)

Relational tables: A relational table is often represented as columns containing one or more parameters with corresponding index columns. Spcifying the different interpretations of a table using format option:

With a single value in a relational table, a singleton parameter is interpreted with the param float value. A.tab contains the table and value is loaded into parameter p:

B A1 A2 A3
1 + - -
2 - + -
3 - - +
import Z.tab format=param: p;

Using a matrix format, the 2-tuple data sets can be represented easily. This matrix format denotes set membership. A relational table is interpreted by the set_array format value as a matrix that defines 2-tuples set with valid tuple (+) and invalid tuple (-).

B A1 A2 A3
1 + - -
2 - + -
3 - - +
import D.tab format=set_array: B;

The data is loaded into set B and following 2-tuples are declared: (‘A1’, 1), (‘A2’, 2), and (‘A3’, 3).

When interpreting parameters with 2-tuple indices, a matrix format is used, where the indices for the rows and columns are different.

I A1 A2 A3

I1 1.3 2.3 3.3

I2 1.4 2.4 3.4

I3 1.5 2.5 3.5

I4 1.6 2.6 3.6

The value is loaded into parameter U (2-dimensional index) using the array format value: import U.tab format=array: A=[X] U;

The data is loaded in a transposed format through transpore_array format value while interpreting the table as a matrix: import U.tab format=transposed_array: A=[X] U;

The index data initialization is do not supported by any of these format values.

Spreadsheets and Relational databases: The usage of the range option is the only way spreadsheets differ from CSV and ASCII text files for importing data. When importing cells from a spreadsheet, the range option indicates the range of cells to be imported. The 1st row of cells defines the table column names for the relational table represented by range of cells.

The data source specification for importing from a relational database is either a filename or data connection string. It may be necessary to specify a username and password since access to a database may be restricted. As an alternative, a data connection string can specify these options within itself.

Pyomo recognizes following database interface packages:

Pyodbc	ODBC interface to databases
pymysql	Python API to MySQL database

Table 3.7: Database Interface Package

- **Automated Functional Optimization**

  Type of problem according to constraints and objective function
  Linear programming problem
  Non-linear programming problem

- **Automated Generation of Quantum Algorithm**

  An ansatz in the context of variational circuits typically refers to a subroutine made up of a series of gates applied to particular wires. Similar to the design of a neural network, this just specifies the basic structure; the variational technique can be used to optimise the types of gates and/or their free parameters.

  There are three different base structures of an ansatz, namely:

  1. Layered gate ansatz

     A layer is a sequence of gates that is repeated. The number of repetitions of a layer forms a hyperparameter of the variational circuit. We can often decompose a layer further into two overall unitaries AA and BB.

  2. Alternating operator ansatz

     The idea of this ansatz is based on analogies to adiabatic quantum computing, in which the system starts in the ground state of AA and adiabatically evolves to the ground state of BB. Quickly alternating (i.e., stroboscopic) applications of AA and BB for very short times Δt can be used as a heuristic to approximate this evolution.

3. Tensor network ansatz

   Amongst the architectures that do not consist of layers, but a single fixed structure, are gate sequences inspired by tensor. The simplest one is a tree architecture that consecutively entangles subsets of qubits.

   Another tensor network is based on matrix product states. The circuit unitaries can be decomposed in different ways, and their size corresponds to the "bond dimension" of the matrix product state — the higher the bond dimension, the more complex the circuit ansatz.

- **Automated Hardware Selection**

  Grover Search – Used for synthesizing the circuit.
  Amplitude Amplification – Used for executing the circuit.

4. EMERGING APPLICATIONS

A. Automated Full Stack Design And Development
B. Automated Education And Creative Development
C. Automated Healthcare and Human Engineering
D. Automated Social Engineering
E. Automated Nature Engineering
F. Automated Resource Generation And Planning
G. Automated Infrastructure and Services
H. Automated Agriculture and Regional Development
I. Automated Industrialization, Ecology and Biodiversity
J. Automated Space Habitat and Floating Cities

Assuming the new automation pipeline:

How User Centric Automation Will Radically Transform Healthcare:

24 x 7 Online Monitoring: Now quantum sensors are succeeding to take the most accurate data from the remotest areas including the processes inside a human cell. These exploit the quantum principles of superposition and entanglement to collect minute changes in the electric and magnetic field of any quantum particle that has motivated the researchers to observe changes in the cellular processes. The idea is that the quantum sensor will capture even a small change in the cell when it gets infected or inflamed or its cellular process disrupts. This leads to important practical applications to observe changes in any single cell or organ or whole body just by wearing a ring made of quantum sensors that will continuously observe the changes in the electromagnetic field and biochemical changes in the body and will notify as soon as there is abrupt change in any parameter like Blood Pressure, SpO2, Heart Rate / Pulse, Stroke, Epilepsy, Behavioral Disorders, Infection (CoVid-19, HIV, STDs etc.), Radiation, Poisoning etc. This way any one will be continuously automatically monitored without any worries of succumbing to deadly diseases or any unforeseen circumstances etc. This technology is leading to revolution in healthcare as immediate cure will stop the development of any disease at the very beginning.

Automated First-Aid: When there is an abrupt change in any health parameter or a sudden change in the biochemical composition of any cell, the quantum sensors in the wearable ring will immediately sense the problem at the root level (i.e. changes in the electric field, magnetic impulse, mechanical properties etc.) and will administer the required precautionary medicine from the ingestible chip besides sending notifications to the family members and the healthcare expert(s) e.g. when the first cell gets infected from any deadly virus like Corona or HIV or STD then the quantum sensors in the wearable ring will immediately detect abrupt changes in the cell and will instruct the ingestible chip to release the medicine to create a shield outside the infected cell to stop spread of the virus. Similarly,

when any person gets exposed to radiation or poisoning or when (s)he faces abrupt changes in blood pressure, heart rate, sugar level etc. then the precautionary medicine will get released from the ingestible chip that will save her / him for 6-8 hours, so (s)he will find enough time to get proper healthcare. This way neither any disease will develop nor the patient will suddenly die.

On demand Diagnosis: When a person wants a detailed diagnosis of any health parameter or organ or whole body, then (s)he will swallow a capsule made of biomolecules, plants, and herbs etc. This capsule will contain the quantum sensor that will bind to the particular cell / organ to be diagnosed. Then, the patient will select the diagnose button from the ring that will connect the quantum states of the ring with that of the quantum sensor bound with the cell. Now the quantum sensor will send the visual data of that particular cell / organ to be diagnosed to the pipeline through the ring. The new inputs will generate the required quantum algorithm to produce the most accurate diagnosis.

Personalized Prescription and Production: Since continuous monitoring and on demand diagnosis will signal the development of any disease at the very beginning, there will be almost no need of medicines. Everybody will remain healthy and disease free by enjoying personalized food, lifestyle, nature care etc. Recommendations for detox & cellular regeneration, personalized nutrition, medicine, yoga, aerobics, dance, sound sleep, electromagnetic field energy, hydro therapy etc. is a multi-objective optimization problem which involves various permutations & combinations in the pipeline to suggest according to the personalized needs and goals of the user. The portable health machines will produce only those items and services that will be finalized by the healthcare expert.

Intensive Care through Automated Bed at Home: Today large number of patients die due to delay in healthcare support but since the patient will be continuously monitored online in the new system, the automated bed at home will immediately provide emergency support like CPR, oxygen support, pre-decided medicine etc. to save life. This way the situation will be under control immediately than worsening due to delay in diagnosis and

un-availability of the doctors. Since the online monitoring of even a quantum particle inside the cell will be visually analyzed, the impact of holistic medicines can be accessed in time. This will motivate the healthcare experts from various therapies to try and test the impact of combined medicines e.g. a proper combination of allopathic medicine, homeopathic medicine, nutrition, magnetic energy, pranic healing etc. If the administered medicines show any kind of negative effect then the visual data will help the healthcare experts to address those issues with appropriate changes (e.g. giving an anti-dote, improving the combination of various medicines or changing the medicine(s) altogether etc.) immediately.

Intensive Critical Care and Near Death Experience at Automated Health Center: Once the intensive care through automated bed at home fails, the patient will be shifted to the automated healthcare center where the most advanced technologies will try to save her / him. Now, some novel initiatives are being experimented to revive the dead person e.g. Dr Sam Parnia succeeds to bring the dead back to life. He argues that the chances of reviving the dead back to life will be more if we automate the following:

- Cardiopulmonary Resuscitation
- Targeted Temperature Management
- Extracorporeal Membrane Oxygenation
- Brain Oximetry
- Prevention of Reperfusion Injury

The automated healthcare center can also try personalized family therapy, ionized water therapy, vibration therapy, sound therapy, mantra therapy etc. Mrit Sanjeevni Vidya, Maha Mrityunjay Jaap, Tantrik Practices etc. can also be personalized by the pipeline and implemented with the most effective machines.

This new type of monitoring, diagnosis, personalized prescription will empower everyone to automatically keep track of health parameters anytime and reduce their dependence on infrastructure centric healthcare systems.

Similarly, **user centric automation** of all the other problems can be done.

5. Conclusion and Future Directions

Conclusion:

- I have proposed a new automation pipeline that will take inputs in the form of text, image, audio, video etc even from an **illiterate user**; will generate the **most efficient quantum algorithm** according to the requirements of the data; and will show the results **visually**.

- I have conceived to ground this new automation pipeline in a new development model valuing labor, creativity and computation to let any user earn 100 % profits from her / his online work.

- Emerging technology for user centric automation will transform everything and will promote the researchers at the helm of affairs.

Future Directions:

- More research is needed to understand the transitioning from Automated Data Cleaning to Automated Data Modeling.

- **Wearable Ring As Input-Output Device for Every Online Facility As A Service:** Since the ring will be just an input-output device to connect the quantum states for

communication and control the instructions by the user, a new type of cloud hardware is required to process data from billions of rings.

- To explore emerging models of computation for a computational theory of everything.

References

[1]. Models of Quantum Computation, http://tph.tuwien.ac.at/~oemer/doc/quprog/node9.html

[2]. Quantum Computation Beyond the Circuit Model by Stephen Paul Jordan, 2008, https://arxiv.org/pdf/0809.2307.pdf

[3]. Quantum Algorithm Implementations for Beginners by Abhijith J. et. al, 2022, ACM Transactions on Quantum Computing, Volume 3, Issue 4, December 2022, Article No.: 18, pp 1–92, https://doi.org/10.1145/3517340

[4]. The Physical Implementation of Quantum Computation by David P. DiVincenzo, 2000, Volume 48, Issue 9-11, September 2000, https://doi.org/10.1002/1521-3978(200009)48:9/11<771::AID-PROP771>3.0.CO;2-E

[5]. A Brief Review of Quantum Algorithms, https://qiskit.org/textbook/preface.html, https://www.allaboutcircuits.com/technical-articles/fundamentals-of-quantum-computing/, https://learn.microsoft.com/en-us/azure/quantum/concepts-grovers

[6]. D. Dong, C. Chen, H. Li and T. -J. Tarn, "Quantum Reinforcement Learning," in *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 38, no. 5, pp. 1207-1220, Oct. 2008, doi: 10.1109/TSMCB.2008.925743.

[7]. Quantum Approximate Optimization Algorithm, https://qiskit.org/textbook/ch-applications/qaoa.html, https://pennylane.ai/qml/demos/tutorial_qaoa_intro.html

[8]. Quantum Neural Networks by Yunseok Kwak, Won Joon Yun, Soyi Jung, and Joongheon Kim, 2021, Conference: 2021 Twelfth International Conference on Ubiquitous and Future Networks (ICUFN), doi:10.1109/ICUFN49451.2021.9528698

[9]. Quantum GAN, Zoufal, C., Lucchi, A. & Woerner, S. Quantum Generative Adversarial Networks for learning and loading random distributions. npj Quantum Inf **5**, 103 (2019), doi: https://doi.org/10.1038/s41534-019-0223-2;

[10]. Quantum Variational Circuits, https://pennylane.ai/qml/glossary/variational_circuit.html

[11]. Recent Advances for Quantum Neural Networks in Generative Learning, https://arxiv.org/pdf/2206.03066.pdf

[12]. Quantum Algorithms for Reinforcement Learning with a Generative Model, http://proceedings.mlr.press/v139/wang21w/wang21w.pdf

[13]. AutoML and AutoRL, https://www.automl.org/, https://medium.com/swlh/8-automl-libraries-to-automate-machine-learning-pipeline-3da0af08f636, https://arxiv.org/pdf/2201.03916.pdf

[14]. List of Awesome Papers in Quantum Machine Learning, https://github.com/artix41/awesome-quantum-ml

[15]. Classiq, https://www.classiq.io/

[16]. OpenAI and DALL E Mini, https://openai.com/

[17]. Blackbox, https://www.useblackbox.io/search

[18]. Minerva, https://ai.googleblog.com/2022/06/minerva-solving-quantitative-reasoning.html

[19]. Hart, William E., Jean-Paul Watson, and David L. Woodruff. "Pyomo: modeling and solving mathematical programs in Python." Mathematical Programming Computation 3, no. 3 (2011): 219-260.

[20]. S. Jannu *et al*., "Energy Efficient Quantum-Informed Ant Colony Optimization Algorithms for Industrial Internet of Things," in *IEEE Transactions on Artificial Intelligence*, 2022, doi: 10.1109/TAI.2022.3220186.