DOI: 10.6977/IJoSI.202504_9(2).0005
P. Narang, P. Mittal, etc./Int. J. Systematic Innovation, 9(2), 47-59 (2025)

Hybrid prediction model by integrating machine learning techniques with MLOps

Poonam Narang[1]*, Pooja Mittal[2], Nisha[3]

[1]Pradhan Mantri Schools for Rising India Government Senior Secondary School Sector 4/7, Gurugram, Haryana, India

[2]Department of Computer Science and Applications, Faculty of Physical Sciences, Maharshi Dayanand University, Rohtak, Haryana, India

[3]Department of Computer Science, Government P.G. College for Women, Rohtak, Haryana, India

*Corresponding author E-mail: poonammdu.rs.dcsa@mdurohtak.ac.in

(Received 26 August 2024; Final version received 26 August 2024; Accepted 24 November 2024)

Abstract

Recent advancements in machine learning (ML) have sparked widespread interest in integrating DevOps capabilities into software and services within the information technology sector. This objective has compelled organizations to revise their development processes. We propose a ML operations model based on meta-ensembling algorithm for gradient boosting regressor with a case study of real estate price prediction. The train and test dataset is loaded with (1460,80) predictive variables, with the sale price as the target variable. The forecasting model is developed using an artificial neural network and a linear logistic regression model, such as LASSO, alongside with the Heroku tool for model deployment. The methodology addresses different steps of data pre-processing, and feature engineering, followed by feature selection, model building, evolution, creating, and calling application programming interfaces for deployment as IaaS, under research, development, and production environment phases. The model is built using the Anaconda Jupyter notebook with various Python libraries and Docker to ensure reproducibility and robustness. To ensure good business value, the performance of the proposed and implemented model is evaluated using different classification metrics, such as area under the curve-ROC for correct assessment measure, alongside accuracy metrics like mean squared error, root mean squared error, and R-squared. Our work serves as a useful reference for building and deploying ML pipeline platforms in practice.

Keywords: DevOps, House-Sale Prediction, Machine Learning, Real Estate Price Prediction

1. Introduction

Increasing advancements in deep learning, along with big data, have fostered the pervasive use of machine learning (ML) and artificial intelligence across various fields (Chen et al., 2014; Lecun et al., 2015). Embedding ML models into different applications makes their development and deployment significantly more complex and challenging than traditional ML implementations. As noted in previous studies (Kumeno, 2019), in feedback loops, the life cycle of ML applications significantly differs from that of traditional methods. Due to the disparities and complexities introduced by computational methods, there is a need for an efficient and reliable approach to developing big data applications, as well as supporting services and infrastructure. The complete life cycle management of ML applications involves multiple stages, infrastructures, artifacts, and channels. Therefore, the increasing adoption of DevOps practices to improve ML processes has gained significant popularity. DevOps practices aim to automate and simplify the integration, testing, acceptance, implementation, and deployment phases, enhancing ML applications to a greater extent (Matsui & Goya, 2020). Another empirical study on machine learning operations (MLOps) identifies and confirms productivity gains following the adoption of DevOps,

such as higher-quality code with continuous sharing, integration, and faster issue resolution (Rzig et al., 2022). Many utilities and key components are also provided by MLOps to manage data storage, access to execution, scheduling, and monitoring of several jobs and pipelines. Developers can use MLOps to configure pipelines or employ an SDK to define ML workflows. These pipelines may include several steps corresponding to different stages of the life cycle, such as data analysis, model training, model evaluation, and model deployment. By leveraging DevOps principles (Bass et al., 2015), ML pipelines benefit from workflow automation and clarification. Continuous integration and continuous delivery (Duvall et al., 2007; Humble & Farley, 2010) bring benefits, such as increased developer productivity and faster code deployment, which can be applied to the iterative development and deployment of ML applications. In addition, continuous training or retraining of models (Google Cloud, 2020) can also be applied to avail new training data and improve the performance deterioration of the model.

MLOps is a concept developed to describe the combination of ML system development and operation through the application of DevOps principles to the life cycle management of ML applications. Aside from ML codes, these frameworks and platforms provide functional components and utilities to help mitigate ongoing maintenance costs resulting from ML system technical debt (Sculley et al., 2015). However, the performance of these platforms is still uncertain in terms of computing resource utilization and the time required to train the model. Further experimentation is necessary to evaluate ML pipeline performance with DevOps integration, or in other words, to assess MLOps using real-world scenarios.

This research applies DevOps practices to ML prediction algorithms for automating and accelerating pipeline deployment. Metrics such as mean square error (MSE), root mean square error (RMSE), and R-squared are used to assess the performance, accuracy, and integrity within the MLOps framework. The main contributions of this research are summarized below:

(i) Proposes an ML-based MLOps model for the deployment of prediction algorithms.
(ii) Review various ML techniques for real estate house-sale price prediction across existing case studies.
(iii) Implemented DevOps automated toolsets to deploy ML pipeline with minimal human intervention.
(iv) Measures performance using various evaluation metrics on the chosen dataset.

The rest of the paper is organized as follows:

(i) Section 2 summarizes previous research relevant to this study.
(ii) Section 3 outlines the use of the proposed MLOps model to evaluate the performance of ML platforms.
(iii) Section 4 discusses the experiment settings, including platform composition, multi-step MLOps pipeline construction, and selected performance metrics.
(iv) Section 5 discusses and analyzes our experimental results.
(v) Section 6 concludes the study.

2. Literature Review

ML applications are evolving from ML programs to developmental ML systems as more computational models are used in software. More than simply applying software engineering principles to the life cycle management of applications and addressing the complexity of maintaining systems with multiple feedback loops, this paper highlights the difficulties in providing extensive and functional infrastructures and platforms to support the development and deployment of ML applications.

2.1. ML Pipeline Platforms

Developing and deploying ML applications entail more than just collecting data, training models, and making predictions. Performing these parts while ignoring proper maintenance can result in significant technical debt (Sculley et al., 2015). To create efficient and reliable ML applications, previous experience and challenges must be considered. For example, ML has been introduced into areas that require high safety, such as autonomous driving and paramedical diagnostics. However, before deploying these applications, ensuring quality and privacy is crucial, necessitating thorough testing and validation of both datasets and trained models.

Creating a workflow from data pre-processing to application runtime monitoring can be time-consuming and error-prone. Automating this process allows developers to focus on ML application development. Furthermore, when new training data becomes available, or model performance deteriorates, computational models must be retrained and redeployed, requiring effective feedback loops from the monitoring system to earlier phases of development. ML platforms such as TFX (Baylor et al., 2017) and ModelOps (Hummer et al., 2019) address the issues raised above. They provide end-to-end life cycle management for ML applications and systems by supplying a set of essential components for tasks such as data pre-processing, model training, model evaluation, and model serving. Designed with pluggable and customizable components, these

platforms aim to provide a generic solution for multiple development scenarios that require different ML tasks. ML workflows can be orchestrated into pipelines that run on these platforms by configuring and integrating different components. While virtual machines and containers are commonly used for training ML models in cloud environments, these platforms also run ML pipelines in hybrid environments. In addition to production-level reliability and scalability, these platforms provide continuous training capabilities, enabling models to adapt to evolving data and increasing update frequency.

2.2. MLOps

ML applications and systems differ from traditional software in many ways, necessitating custom DevOps for ML features. Traditional software systems contain fewer model artifacts and more data processing steps and must deal with more complex relationships between these artifacts. The training outcomes must be traceable, robust, and reproducible due to the use of computational models and their experimental nature (Olorisade et al., 2017). MLOps is an ML engineering practice that applies DevOps principles to ML systems, unifying the development and operation of ML systems. In terms of continuous integration, additional test procedures, such as data and model validations, are introduced alongside traditional unit and integration tests. Processed datasets and trained models are automatically and continuously delivered from data and deep learning scientists to ML system engineers through continuous deployment. Continuous testing dictates that the arrival of new data and the deterioration of model performance necessitate model retraining or performance improvement through online methods (Google Cloud, 2020; Wikipedia, 2020). In a similar context, Ruf et al. (2021) have demystified the process of selecting among numerous existing open-source tools. They also acknowledge that as more tools become available for different operational phases, defining responsibilities and requirements becomes increasingly complex. With this goal in mind, the authors investigated and clearly defined MLOps technologies and tools based on carefully chosen requirements, including input data, model performance, and system quality metrics. Frameworks for MLOps theory have also been developed in various research works (John et al., 2021; Makinen et al., 2021; Marrero & Astudillo, 2021; Subramanya et al., 2022) and applied to forecast or predict various real-world applications, such as the generation of electricity bills. Their work further emphasizes the importance of using standardized frameworks or models for generalized applications.

2.3. ML and House-Sale Predictions

Previously, the real estate industry was not recognized as an advanced industrial category. However, with the advancement of ICT and its integration with numerous financial markets and investments, the real estate industry has become more dynamic (Kang et al., 2020). Many researchers have examined the performance of various ML algorithms on various real-world datasets to predict real estate or house sales. In fact, ML is increasingly being used for large-scale real estate appraisals, followed by automated valuation models. Data from real estate listings is collected and used in mass appraisals to estimate property values, ensuring that appraisals are carried out consistently and impartially (Mora-Garcia et al., 2022). In a similar context, several renowned researchers (Fan et al., 2018; Jui et al., 2020; Wang & Li, 2019) have proposed different proposals and algorithms for an innovative real estate valuation approach. These studies also address the limitations of correlation coefficients in traditional approaches. Other studies have attempted to identify the most effective ML algorithms for predicting house prices and analyzing the impact of the coronavirus disease 2019 pandemic on house prices using different datasets, such as Spanish city, Shenzhen (China), and Dhaka (Bangladesh) (Cheung et al., 2021; Kaynak et al., 2021; Neloy et al., 2019; Pai & Wang, 2020). The algorithms random forest (RF), extra trees regressor, gradient boosting regressor, support vector regressor, multilayer perceptron neural network, and k-nearest neighbors (kNN) were used. Their findings indicated that RF and ETR algorithms outperformed other algorithms in terms of predictive performance.

2.4. Motivation and Social Relevance

House-sale prediction is critical for enhancing real estate efficiency. House prices are determined, as previously stated, by calculating the acquisition and selling prices within a neighborhood. As a result, the house-sale prediction model plays an important role in bridging the information gap and improving real estate efficiency. With the proposed model, we can more accurately predict prices. Prediction systems have become increasingly important in our lives with the rise of platforms, such as YouTube, Amazon, and Netflix, over the last few decades. These systems are now unavoidable in our daily online experiences, from e-commerce (suggesting products that may be of interest to buyers) to online advertising (suggesting relevant content based on user preferences).

3. Proposed MLOps Model

The proposed working model is segmented into three different parts: research environment,

development environment, and production environment, as shown in Fig. 1.

To investigate and estimate the performance of ML pipelines on a specific platform, we must first construct an ML platform based on previous work. Meanwhile, we want the platform to support continuous training by automatically retraining models when specific events occur on a regular basis, such as changes to the ML algorithm code. Finally, we will develop an efficient ML pipeline that can be run on this platform.

3.1. Research Environment

In the research environment, building of ML pipelines is the major step, which includes the following steps:

(i) Data analytics
(ii) Feature engineering
(iii) Feature selection
(iv) Model training
(v) Obtaining predictions/scoring

As shown in Fig. 2, the pipeline starts with data collection and analytics and ends with different predictions for the underlying dataset.

3.2. Development Environment

This phase creates an application programming interface (API) and makes calls to the API. It is also responsible for correlating the research and production models to produce the same outcome when given the same data.

3.3. Production Environment

After building an ML model using data science on Jupyter notebook, the development code is

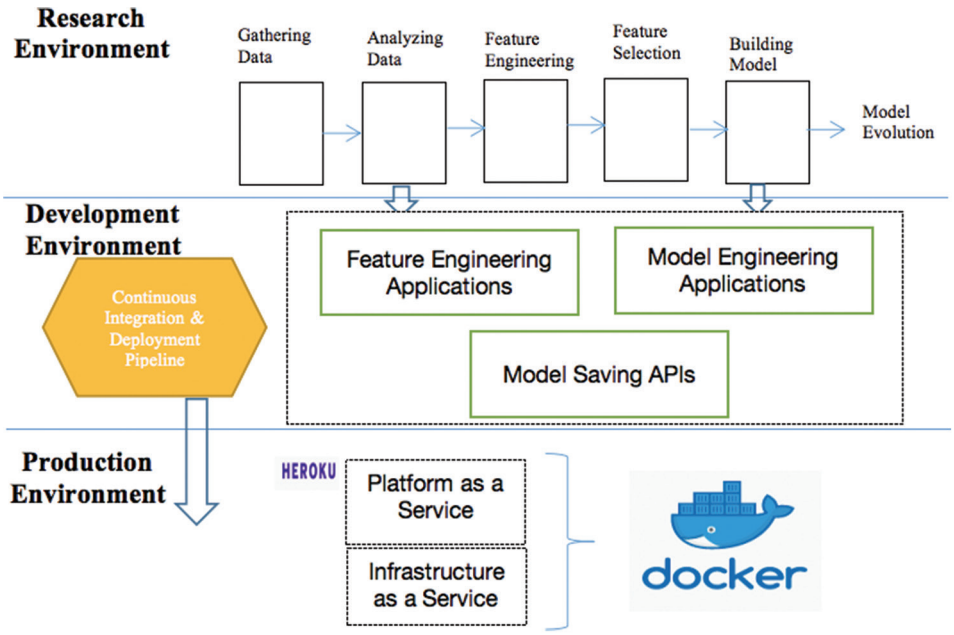


Fig. 1. Proposed machine learning operations model for deployment of machine learning algorithm

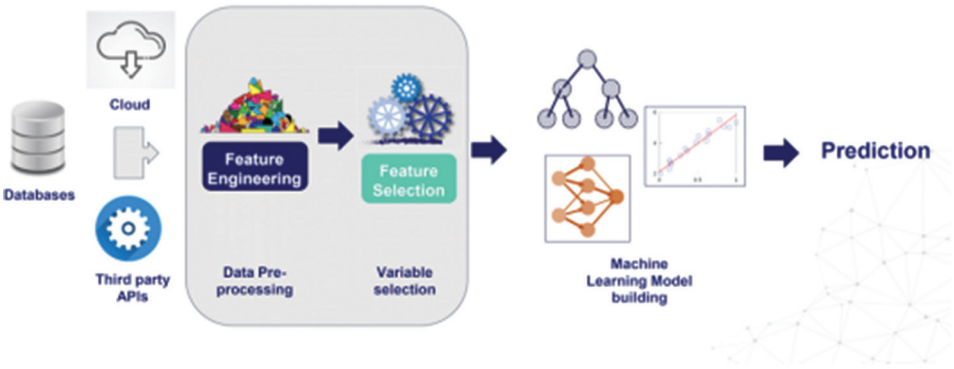


Fig. 2. Machine learning pipeline for making predictions

transformed into code that can be used in production. In the deployment of ML models (Fig. 4), we take our models from the research stage (Jupyter) to a fully integrated API that can be called live to make predictions on real-time data.

With continuous integration and various deployment solutions, such as platform or infrastructure as a service, Docker is also used to ensure model reproducibility and robustness. The model is built and deployed with Python as our main language.

4. Detailed Architecture of the Model

ML model deployment refers to the process of making models available in production environments where they can provide predictions to other software systems. Models begin adding value and making predictions after they are deployed to production, so deployment is an important step. However, deploying ML models is difficult. This section describes the architecture or environments in detail.

4.1. Data Collection

Data collection mainly refers to the methods or procedures of data acquisition from different resources. Real-time datasets may be collected from public repositories such as Kaggle, UCI, and Psyionet. The dataset under consideration is taken from the house price dataset from Kabul through a publicly available online Kaggle data repository.

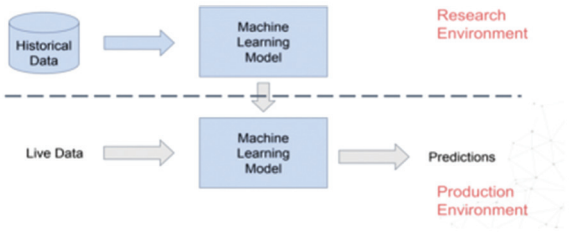


Fig. 3. Development environment to ensure reproducibility of outcome

4.2. Data Analytics/Pre-processing

A typical ML pipeline involves steps of gathering the data, typically coming from different areas of the business or different data sources, then transforming that data in various forms to tackle the quality of the data or to create new features. This first step of data gathering makes the data available to the data scientists so they can go ahead and build the ML models. In data analytics or pre-processing, we need a good understanding of what the data are telling us. It is a good practice to know the data well, to get familiar with the variables, to know how the variables are related to each other, and to know what we want to predict. If this was a supervised case, we need to know what variables we can use. Surely, there are regulations in your business and which variables we cannot use. Once we have analyzed our data, the next step is feature engineering after data analysis, which you have gained a good understanding of whether we can use the variables as they are or if we need to transform them before passing them onto an ML algorithm.

4.3. Feature Engineering

During feature engineering, we transform the variables to make them ready to be utilized in an ML model. There are a variety of problems that we can find in the variables in our datasets as shown in Fig. 5 below.

As described in Fig. 5, one of the problems is missing data, meaning an absence of values for certain observations within a variable. There could be a variety of reasons why data could be missing, a value can be lost or not stored properly during data storage, or the value does not exist. Other problems may include the presence of rare labels in categorical variables, the distribution of the variables for numerical variables, and the presence of outliers. These problems, especially the outliers, may affect certain ML models or linear regressions, and this tends to cause over-fitting in a bad

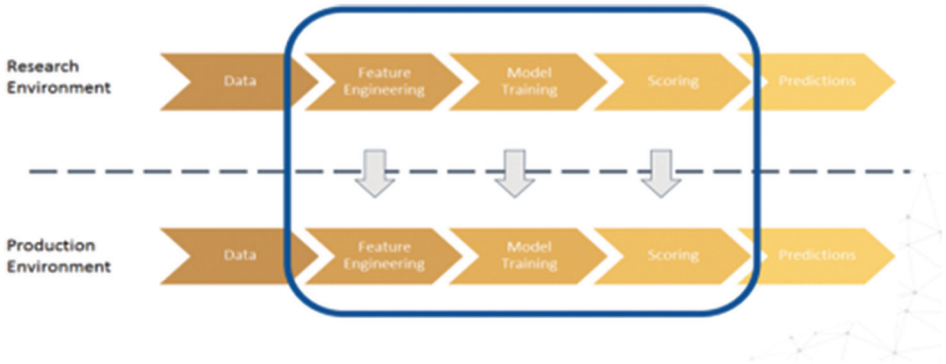


Fig. 4. Deployment of machine learning pipeline in a production environment

generalization. Finally, the magnitude of the features also affects the performance of the working model.

For example, if we were trying to predict house price and we have one variable, which is the area in terms of square kilometers, and the other variable is the number of rooms that vary from 1 to 10. Hence, in a linear model, the variable that takes the larger values would have a predominant role over the house price. In this case, the area variable would be more important to determine the price of the house. However, we know that the number of rooms also plays a role. As in the linear model, Y, which also supports vector machines and neural networks, is supposed to converge faster and find the optimal hyperplane faster when using features with a similar scale. Furthermore, all the distance-based algorithms are sensitive to the scale of the features.

Techniques that we can use to tackle each of the problems include variable transformations, feature extraction, and creating new features, as shown in Fig. 6 below.

Fig. 5. Different aspects of feature engineering

Fig. 6. Feature engineering techniques

4.4. Feature Selection

After feature engineering, we can select the features that we want to use in the ML models. Feature selection refers to algorithms or procedures that will allow us to find the best subset of features from all the variables present in our dataset. This is a process to identify the most predictive features at the beginning of the feature selection process. We start with the entire dataset, with all the variables, and by the end of the feature selection process, we end up with a smaller number of features, which are typically the most predictive ones. There are several reasons why we build models using fewer features. One of the reasons is that these are easier to put into production, as we need to ship smaller, Jason, messages between the business systems and the model. Second, we need to write less code to pre-process those features, and we also need to write less code to handle potential errors, and then they will take the predictions out of our systems. There are three umbrella terms under which we group the different feature selection algorithms. One group corresponds to the embedded methods, another group to the rapid methods, and then we have the filter methods.

We can make the feature selection part of the pipeline, but the issue is better resolved if we select the features ahead of building the pipeline that we want to deploy and then make the list of the selected features part of the pipeline that we want to deploy.

4.5. Model Training

After feature selection, we train the models to build our ML algorithms. There are several models that we can build. We can build, for example, linear models such as linear logistic regressions or MARS, decision trees-based algorithms such as RF and gradient-boosted trees, and neural networks for super-biased models. We can also build clustering algorithms or recommender systems. This research work builds the MLOps forecasting model developed with ANN and a linear logistic regression model, LASSO.

4.6. Making Predictions/Score Values

After model training, the next crucial step comes under the research environment is obtaining predictions and evaluating the model performance. We need to deploy the entire ML pipeline and not just the ML model because what we need to have in the production environment is a complete sequence of steps that take in the raw data and outputs a final prediction. Hence, when we passed the pre-processed data to our models, we were able to get the predictions that they made. We then need to evaluate the predictions that these models make. To make sure that the models bring good business value, we evaluate the performance using different metrics depending on the project; for example, we measure the area under the curve-RAC, which gives us an indication of how many times the model makes a good assessment versus how many times the model makes the wrong assessment. We also measure the accuracy, MSE, RMSE, and R2 errors for linear models.

4.7. ML Algorithms

Two ML algorithms were pitted against each other in this study to see which one was better at predicting housing prices. Baldominos et al. (2018)

compared four ML algorithms for housing prices in a similar study. They discovered that the RF regression algorithm predicted the least error, followed by the kNN regression algorithm, in their research. kNN regression and Artificial Neural Networks are proposed as methods for predicting house prices in Oxenstierna's (2017) study. Because both reports look at the performance of kNN regression, the algorithm will be looked at in this report as well. RF regression will be included in the study and compared to the kNN regression algorithm because Baldominos et al. (2018) discovered that it has the lowest error for predicting house prices and Oxenstierna (2017) mentions it as relevant for future work.

4.8. kNN Regression

The kNN algorithm is a non-parametric method for solving classification and regression problems. The algorithm assumes that any item in the dataset with similar values for other features will have similar values for the prediction target. The target variable in our case is the house price, which is predicted by the k number of neighbors with the most similar features.

The information can be visualized as points in an n-dimensional Cartesian space. In the two-dimensional case, for example, we have two features with values represented as points on a plane. Fig. 7 depicts the case when k = 3.

The three nearest samples are identified by lines from the test sample. We are concerned with determining a numerical value for the unknown variable because we are dealing with regression. Using the mean of the nearest neighbor is a straightforward method. However, some of those k points may be much further apart than others. To combat this, weights can be assigned to each neighbor based on some function based on distance. One method is to weigh them in inverse proportion to the distance. In many cases, the inverse distance weighted average approach outperforms uniform weights, i.e., no weights. In practice, the number of neighbors to include in the calculation (i.e., the size of k) is determined by trial and error, comparing prediction errors for various values of k.

4.9. RF Regression

RF is an algorithm that can be used for classification as well as regression. RF models are built by assembling a set of decision trees based on training data. Instead of using a single tree to predict the target value, the RF algorithm uses the average prediction of a group of trees. The decision trees are built by fitting to randomly generated groups of rows and columns in the training data. This method is known as bagging,

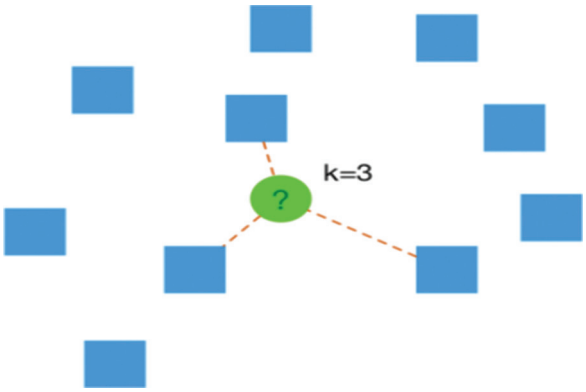
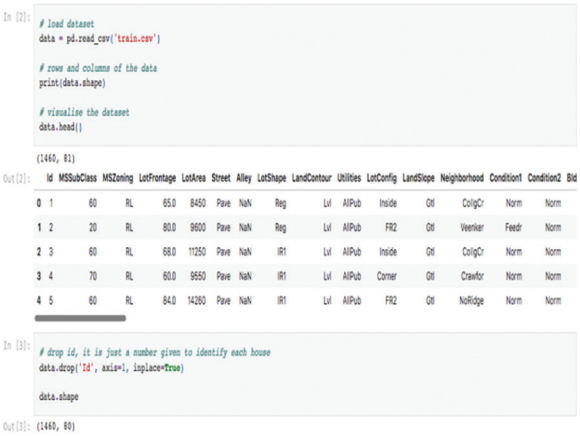


Fig. 7. The k-nearest neighbor algorithm with k = 3



```
In [2]: # load dataset
        data = pd.read_csv('train.csv')

        # rows and columns of the data
        print(data.shape)

        # visualise the dataset
        data.head()
```

```
(1460, 81)
```

Out[2]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	Bld
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	Veenker	Feedr	Norm	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	Norm	Norm	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	Norm	Norm	

```
In [3]: # drop id, it is just a number given to identify each house
        data.drop('Id', axis=1, inplace=True)

        data.shape
```

```
Out[3]: (1460, 80)
```

Fig. 8. Dataset view for house-sale price prediction in Kaggle

and it results in less bias because each tree is built at random on different parts of the input. The method of averaging decision tree predictions reduces over-fitting that can occur when using single decision trees.

To determine which ML method is best for the house price problem, the prediction accuracy of the algorithms kNN and RF were compared. Instead of writing algorithms from scratch, algorithms from the scikit-learn library were used in this study. It is a cutting-edge Python library that is part of the scikit suite of scientific toolkit. Pandas, the data analysis library from "Our Python," was also used. The dataset was pre-processed and cleaned before comparing the algorithms so that the algorithms could use it as input. Furthermore, a method for evaluating the data has been established, and finally, the ML algorithms for prediction using the cleaned dataset have been tested with different values for relevant hyper-parameters.

5. Results and Discussion

This section details the experimental setup for the prediction of house-sale prices in Kabul using a linear regression model, followed by implementation

and deployment of the proposed model in terms of making predictions.

5.1. Dataset

As described in Fig. 2, gathering data coming from different sources undergoes several stages as pre-processing of data. The current research work is based on a house-sale price dataset for Kabul obtained from a publicly available online Kaggle repository (https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data).

In general, ML algorithms are designed to accept only numerical data as input. More than half of the columns in the Housing dataset are non-numerical and must be encoded, which is done with one-hot encoding and labeling in this case. Data pre-processing is in high demand because only by providing accurate and error-free data to our model will the model be able to provide precise estimates that are very close to the actual value. For the sake of model accuracy and over-fitting, we remove null values, perform an overview of the dataset, and remove unnecessary data columns (independent attributes) in data pre-processing and cleaning. Several columns also have some empty values that have been handled in various ways. These generic methods of normalizing data include encoding categorical data and detecting missing values, outliers, temporal variables, discrete variables, and continuous variables. For example, outliers are errors in the collected entries that occur as a result of the manual collection of data through web scraping, such as null or blank values, human errors, or impractical values. To compensate for these errors, we must pre-process the

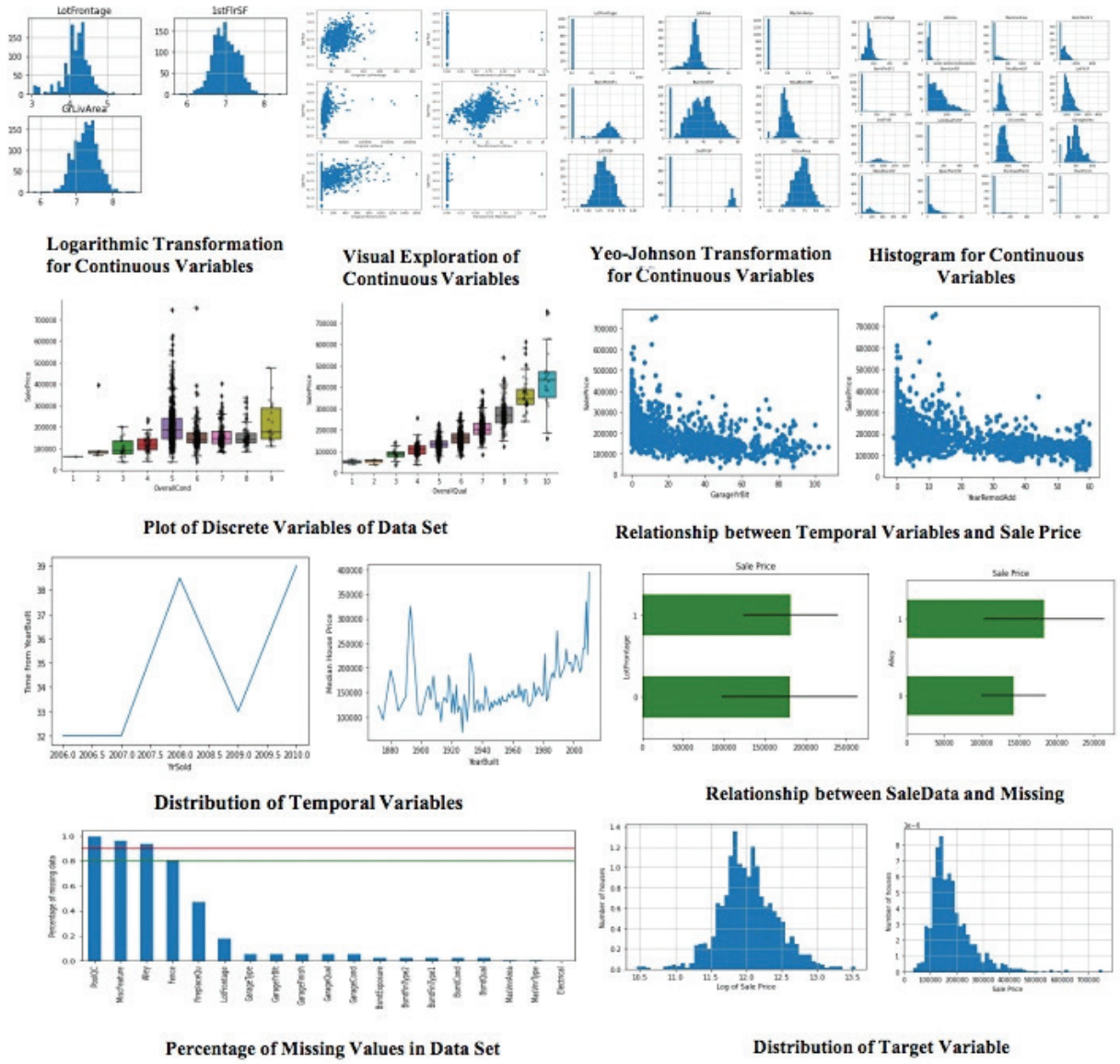


Fig. 9. Data analysis for different types of variables present in the dataset

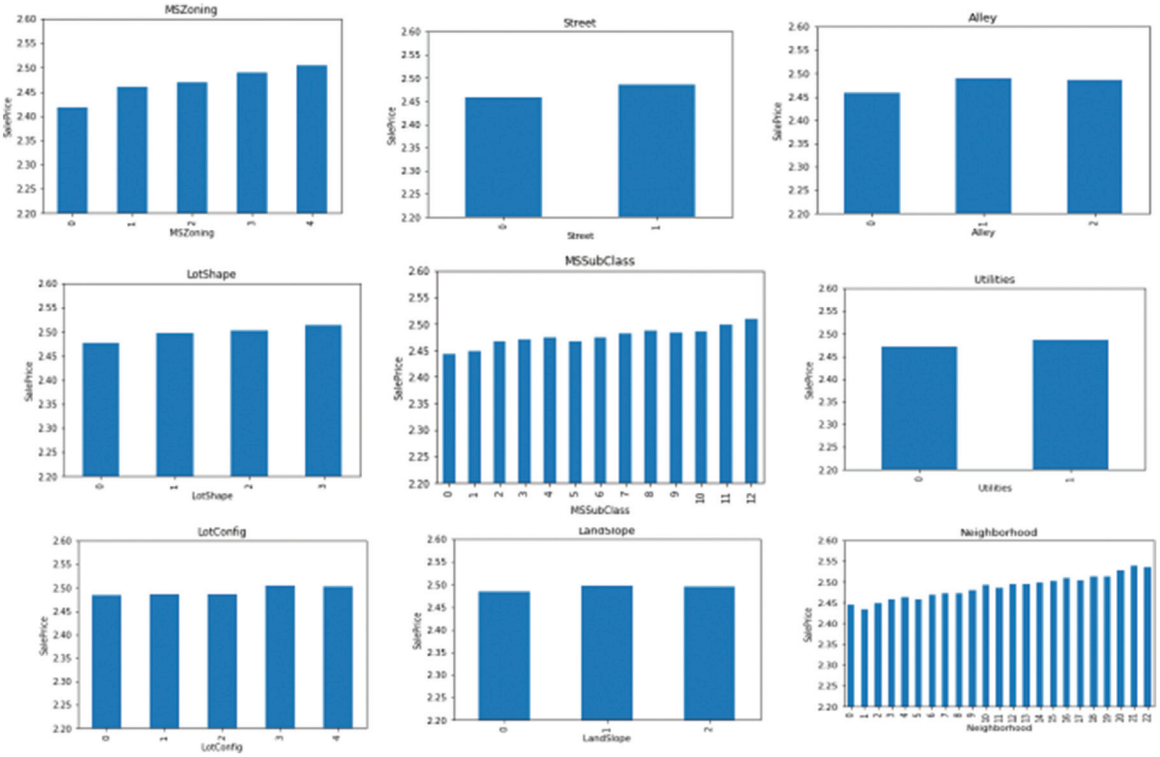


Fig. 10. Plot of monotonic variables versus target during feature engineering

```
In [6]:
# let's visualise those features that were selected.
# (selected features marked with True)

sel_.get_support()
```

```
Out[6]: array([ True,  True,  True, False, False, False,  True,  True, False,
                True, False,  True, False, False, False, False,  True,  True,
               False,  True,  True, False,  True, False, False, False,  True,
               False,  True,  True, False,  True,  True, False, False, False,
               False, False, False,  True,  True, False,  True,  True, False,
                True,  True, False, False,  True, False, False,  True,  True,
                True,  True,  True, False, False,  True,  True,  True, False,
               False,  True,  True, False, False, False,  True, False, False,
               False, False, False, False, False,  True, False, False, False])
```

```
In [7]:
# let's print the number of total and selected features

# this is how we can make a list of the selected features
selected_feats = X_train.columns[(sel_.get_support())]

# let's print some stats
print('total features: {}'.format((X_train.shape[1])))
print('selected features: {}'.format(len(selected_feats)))
print('features with coefficients shrank to zero: {}'.format(
    np.sum(sel_.estimator_.coef_ == 0)))
```

```
total features: 81
selected features: 36
features with coefficients shrank to zero: 45
```

```
In [8]:
# print the selected features
selected_feats
```

```
Out[8]: Index(['MSSubClass', 'MSZoning', 'LotFrontage', 'LotShape', 'LandContour',
       'LotConfig', 'Neighborhood', 'OverallQual', 'OverallCond',
       'YearRemodAdd', 'RoofStyle', 'Exterior1st', 'ExterQual', 'Foundation',
       'BsmtQual', 'BsmtExposure', 'BsmtFinType1', 'HeatingQC', 'CentralAir',
       '1stFlrSF', '2ndFlrSF', 'GrLivArea', 'BsmtFullBath', 'HalfBath',
       'KitchenQual', 'TotRmsAbvGrd', 'Functional', 'Fireplaces',
       'FireplaceQu', 'GarageFinish', 'GarageCars', 'GarageArea', 'PavedDrive',
       'WoodDeckSF', 'ScreenPorch', 'SaleCondition'],
      dtype='object')
```

```
In [9]:
pd.Series(selected_feats).to_csv('selected_features.csv', index=False)
```

Fig. 11. Feature selection code implementation

```
In [7]:
# evaluate the model:
# ===================

# remember that we log transformed the output (SalePrice)
# in our feature engineering notebook (step 2).

# In order to get the true performance of the Lasso
# we need to transform both the target and the predictions
# back to the original house prices values.

# We will evaluate performance using the mean squared error and
# the root of the mean squared error and r2

# make predictions for train set
pred = lin_model.predict(X_train)

# determine mse, rmse and r2
print('train mse: {}'.format(int(
    mean_squared_error(np.exp(y_train), np.exp(pred)))))
print('train rmse: {}'.format(int(
    mean_squared_error(np.exp(y_train), np.exp(pred), squared=False))))
print('train r2: {}'.format(
    r2_score(np.exp(y_train), np.exp(pred))))
print()

# make predictions for test set
pred = lin_model.predict(X_test)

# determine mse, rmse and r2
print('test mse: {}'.format(int(
    mean_squared_error(np.exp(y_test), np.exp(pred)))))
print('test rmse: {}'.format(int(
    mean_squared_error(np.exp(y_test), np.exp(pred), squared=False))))
print('test r2: {}'.format(
    r2_score(np.exp(y_test), np.exp(pred))))
print()

print('Average house price: ', int(np.exp(y_train).median()))

train mse: 781396538
train rmse: 27953
train r2: 0.8748530463468015

test mse: 1060767982
test rmse: 32569
test r2: 0.8456417073258413

Average house price:  163000
```

Fig. 12. LASSO model implementation code

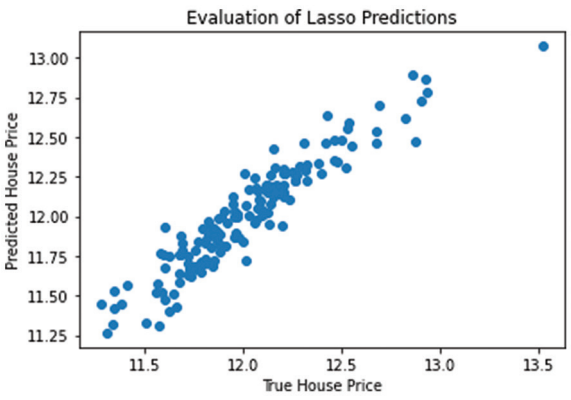


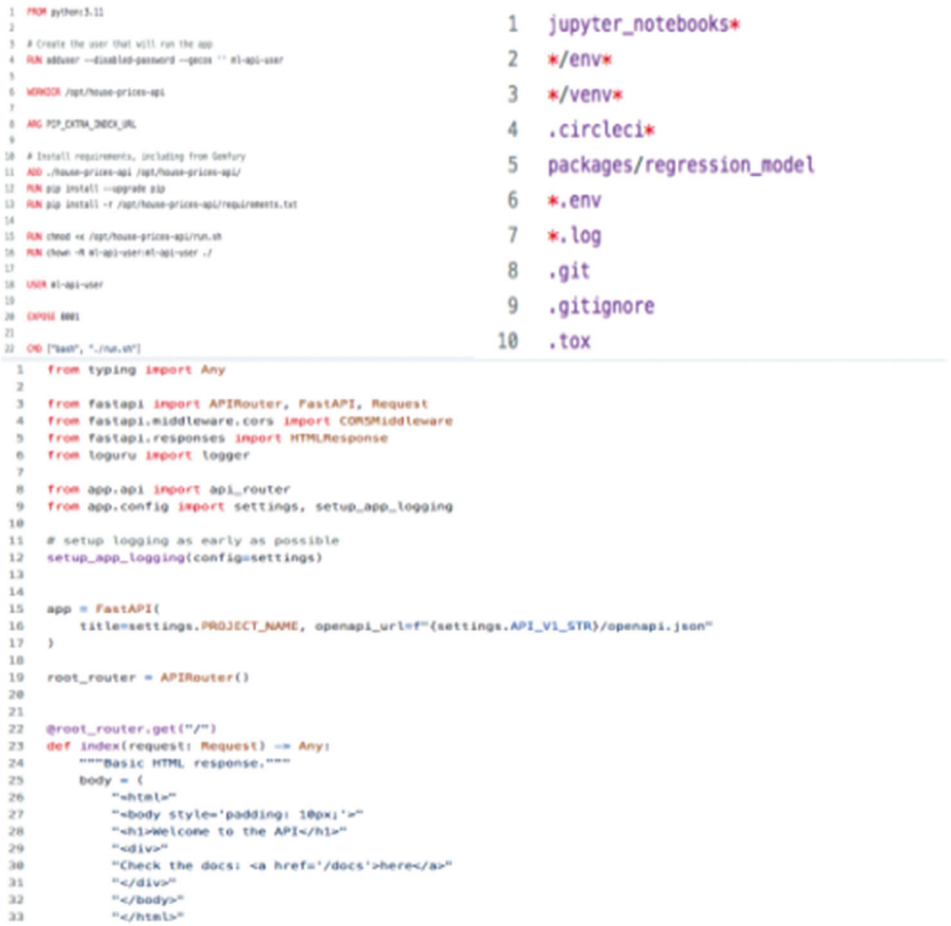
Fig. 13. LASSO model evaluations for the prediction of house-sale price

data and remove the clutter values. All these different types of variables are plotted in graphical notation to check their evidence and are shown in Fig. 9 before transformation.

Fig. 9 shows the graphical representations for different types of variables in the house-sale price prediction dataset.

5.2. Feature Engineering of the Dataset

After the data have been cleaned and making it free from outliers, feature engineering and exploratory data analysis have to be done. Fig. 10 shows the histograms for cleaned data after feature engineering.



```
jupyter_notebooks*
*/env*
*/venv*
.circleci*
packages/regression_model
*.env
*.log
.git
.gitignore
.tox
```

```python
from typing import Any

from fastapi import APIRouter, FastAPI, Request
from fastapi.middleware.cors import CORSMiddleware
from fastapi.responses import HTMLResponse
from loguru import logger

from app.api import api_router
from app.config import settings, setup_app_logging

# setup logging as early as possible
setup_app_logging(config=settings)


app = FastAPI(
    title=settings.PROJECT_NAME, openapi_url=f"{settings.API_V1_STR}/openapi.json"
)

root_router = APIRouter()


@root_router.get("/")
def index(request: Request) -> Any:
    """Basic HTML response."""
    body = (
        "<html>"
        "<body style='padding: 10px;'>"
        "<h1>Welcome to the API</h1>"
        "<div>"
        "Check the docs: <a href='/docs'>here</a>"
        "</div>"
        "</body>"
        "</html>"
```

Fig. 14. Python code for deploying machine learning model to production

5.3. Feature Selection after Feature Engineering

Because having irrelevant features in our data can reduce the model's accuracy, we use feature selection to automatically or manually select the features that contribute the most to the prediction variable. To select features for the final model, filter methods, such as constant feature elimination, quasi-constant feature elimination, duplicate feature elimination, fisher score, univariate method, mutual information, and correlation, are used. Wrapper methods such as step-forward selection, step-backward selection, and exhaustive search are examples of such methods. Methods such as decision tree-derived importance and regression coefficients are embedded.

5.4. Fitting the ML Model

The data are then divided into training and testing sets to classify the best-fitting ML model. The standard 80-20 split ratio is used: 80% of the data is considered a training set, and 20% is considered a testing set. Scikit-Learn must be imported before the model can be implemented. It is a Python library that provides ML algorithms for implementation as well as many other modeling features. We are using the supervised regularization method, LASSO, to perform precise price estimation. Our model will be the one with the lowest error and the closest value prediction. After setting the seed for the model, the next step comes to implement the model to predict different accuracy measures, such as MSE, RMSE, and R2 and to give good predictions for our hose-sale price dataset. Fig. 11 shows the code that we used for feature selection.

Fig. 13 shows the best-fit curve under the LASSO model for the price prediction dataset.

Based on the observations above, it is clear that linear regression produces the most precise results and has been chosen as the predictive model for house-sale prediction. The model is ready for use as an analytic tool for both real estate business managers and buyers.

5.5. Deployment of the Model

Once implemented, the model predicts the price of the property (house) in that specific location, Kabul, as selected in our dataset. Next comes deploying the model with the Docker container framework, in which

the user can enter the desired values and our model predicts the output. This is made possible using the Python package Heroku and Docker to create an API. To build the web application and connect the Model to it, we must first extract our model into pickle and.json files and design a web page using HTML, CSS, and JavaScript. The model is now ready to be displayed and predicted on the web application.

6. Conclusion and Future Scope

The LASSO-supervised regularization ML model has proven to be an effective method for determining the best-fitting algorithm for a model. This linear regression algorithm provides a very accurate estimation of house prices. It provides much more accurate estimations for various locations. Furthermore, linear regression provides nearly accurate predictions based on the confusion matrix. Linear regression fits our dataset and performs well. Deployment of the model after implementation helps to predict the value automatically for different datasets. In the future, an appealing and interactive graphical user interface can be created to integrate into any real estate sale website, where sellers can provide details and houses for sale and buyers can contact based on the information provided on the website. To make things easier for the user, there could be a recommending system that recommends real estate properties based on the predicted price. The current dataset only includes a few locations in Kabul. Expanding it to other Indian cities and states is the long-term goal. Google Maps can be included to make the system even more informative and user-friendly. This will display the neighborhood amenities, such as hospitals and schools within a 1 km radius of the given location. This can also be factored into predictions because the presence of such factors raises the value of real estate property.

References

Baldominos, A., Blanco, I., Moreno, A.J., Iturrarte, R., Bernárdez, Ó., & Afonso, C. (2018). Identifying real estate opportunities using machine learning. *Applied Sciences*, 8(11), 2321. https://doi.org/10.3390/app8112321

Bass, L., Weber, I., & Zhu, L. (2015). *DevOps: A Software Architect's Perspective.* Addison-Wesley Professional, Boston.

Baylor, D., Breck, E., Cheng, H.T., Fiedel, N., Foo, C.Y., Haque, Z., Haykal, S., Ispir, M., Jain, V., Koc, L., & Koo, C.Y. (2017). Tfx: A Tensorflow-Based Production-Scale Machine Learning Platform. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1387–1395.

Chen, M., Mao, S., & Liu, Y. (2014). Big data: A survey. *Mobile Networks and Applications*, 19, 171–209.

Cheung, K.S., Yiu, C.Y., & Xiong, C. (2021). Housing market in the time of pandemic: A price gradient analysis from the COVID-19 epicentre in China. *Journal of Risk and Financial Management*, 14(3), 108.

Duvall, P.M., Matyas, S., & Glover, A. (2007). *Continuous Integration: Improving Software Quality and Reducing Risk.* United Kingdom: Pearson Education.

Fan, C., Cui, Z., & Zhong, X. (2018). House Prices Prediction with Machine Learning Algorithms. In: *Proceedings of the 2018 10th International Conference on Machine Learning and Computing, February 26–28, 2018*, Macau, China, pp. 6–10.

Google Cloud. (2020). *MLOps: Continuous Delivery and Automation Pipelines in Machine Learning.* Available from: https://cloud.google.com/solutions/machine-learning/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation.* Pearson Education, United Kingdom.

Hummer, W., Muthusamy, V., Rausch, T., Dube, P., El Maghraoui, K., Murthi, A., & Oum, P. (2019). Modelops: Cloud-Based Lifecycle Management for Reliable and Trusted Ai. In: *2019 IEEE International Conference on Cloud Engineering (IC2E), June 24–27, 2016*, Prague, Czech Republic, pp. 113–120.

John, M.M., Olsson, H.H., & Bosch, J. (2021). Towards MLOps: A Framework and Maturity Model. In: *Proceedings of the 2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), September 1–3, 2021*, Palermo, Italy, pp. 334–341.

Jui, J.J., Imran Molla, M.M., Bari, B.S., Rashid, M., & Hasan, M.J. (2020). Flat Price Prediction Using Linear and Random Forest Regression Based on Machine Learning Techniques. Embracing Industry 4.0. Embracing Industry 4.0: Selected Articles from MUCET 2019. Vol. 678. Springer, Singapore, pp. 205–217.

Kang, J., Lee, H.J., Jeong, S.H., Lee, H.S., & Oh, K.J. (2020). Developing a forecasting model for real estate auction prices using artificial intelligence. *Sustainability*, 12(7), 2899. https://doi.org/10.3390/su12072899

Kaynak, S., Ekinci, A., & Kaya, H.F. (2021). The effect of COVID-19 pandemic on residential real estate prices: Turkish case. *Quantitative Finance and Economics*, 5, 623–639. https://doi.org/10.3934/QFE.2021028

Kumeno, F. (2019). Software engineering challenges for machine learning applications: A literature review. *Intelligent Decision Technologies*, 13(4), 463–476. https://doi.org/10.3233/IDT-190160

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444. https://doi.org/10.1038/nature14539

Makinen, S., Skogstrom, H., Laaksonen, E., & Mikkonen, T. (2021). Who needs MLOps: What Data Scientists Seek to Accomplish and how can MLOps Help? In: *Proceedings of the 2021 IEEE/ACM 1st Workshop on AI Engineering—Software Engineering for AI (WAIN), May 30–31, 2021, Madrid, Spain*.

Marrero, L., & Astudillo, H. (2021). *DevOps-RAF: An Assessment Framework to Measure DevOps Readiness in Software Organizations. Proceedings of the 2021 40th International Conference of the Chilean Computer Science Society (SCCC), November 15–19, 2021,* La Serena, Chile, pp. 1–8.

Matsui, B., & Goya, D. (2020). *Application of DevOps in the Improvement of Machine Learning Processes. Conference Paper*.

Mora-Garcia, R.T., Cespedes-Lopez, M.F., & Perez-Sanchez, V.R. (2022). Housing price prediction using machine learning algorithms in COVID-19 times. *Land*, 11(11), 2100. https://doi.org/10.3390/land11112100

Neloy, A.A., Haque, H.S., & Ul Islam, M.M. (2019). Ensemble Learning Based Rental Apartment Price Prediction Model by Categorical Features Factoring. In: *Proceedings of the 2019 11th International Conference on Machine Learning and Computing, February 22–24, 2019*, Zhuhai, China, pp. 350–356.

Olorisade, B.K., Brereton, P., & Andras, P. (2017). *Reproducibility in Machine Learning-based Studies: An Example of Text Mining*. Reproducibility in Machine Learning, Autralia.

Oxenstierna, J. (2017). *Predicting House Prices using Ensemble Learning with Cluster Aggregations.* Bachelor thesis, Uppsala University, Sweden.

Pai, P.F., & Wang, W.C. (2020). Using Machine Learning Models and Actual Transaction Data for Predicting Real Estate Prices. *Applied Sciences*, 10, 5832. https://doi.org/10.3390/app10175832

Ruf, P., Madan, M., Reich, C., & Ould-Abdeslam, D. (2021). Demystifying mlops and presenting a recipe for the selection of open-source tools. *Applied Sciences*, 11(19), 8861. https://doi.org/10.3390/app11198861

Rzig, D.E., Hassan, F., & Kessentini, M. (2022). An empirical study on ML DevOps adoption trends, efforts, and benefits analysis. *Information and Software Technology*, 152, 107037. https://doi.org/10.1016/j.infsof.2022.107037

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.F., & Dennison, D. (2015). Hidden technical debt in machine learning systems. *Advances in Neural Information Processing Systems*, 28, 2503–2511.

Subramanya, R., Sierla, S., & Vyatkin, V. (2022). From DevOps to MLOps: Overview and application to electricity market forecasting. *Applied Sciences*, 12(19), 9851. https://doi.org/10.3390/app12199851

Wang, D., & Li, V.J. (2019). Mass appraisal models of real estate in the 21st century: A systematic literature review. *Sustainability*, 11, 7006. https://doi.org/10.3390/su11247006

Wikipedia. (2020). *Online Machine Learning*. Available from: https://en.wikipedia.org/wiki/online_machine_learning