HelloBetter

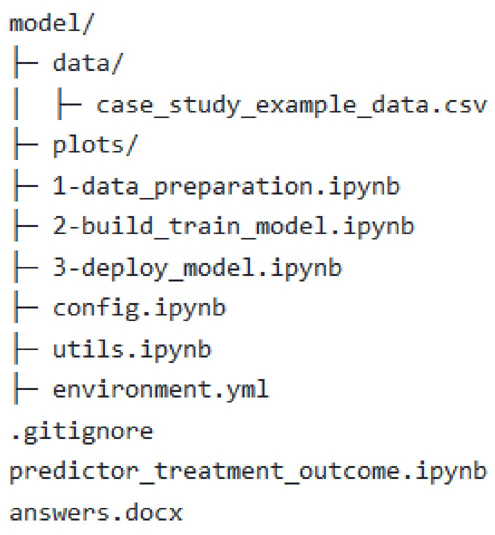
Case Study Challenge

Machine Learning Engineer / Data Scientist

Oael Chabir
11-15-2023

Repo structure

Link of the repo: https://github.com/chabirOael/HelloBetter_Challenge



```
model/
├ data/
|  ├ case_study_example_data.csv
├ plots/
├ 1-data_preparation.ipynb
├ 2-build_train_model.ipynb
├ 3-deploy_model.ipynb
├ config.ipynb
├ utils.ipynb
├ environment.yml
.gitignore
predictor_treatment_outcome.ipynb
answers.docx
```

- **model/**: a directory containing all model pipeline
  - ➢ Data preparation **(1-data_preparation.ipynb)**
  - ➢ Model training **(2-build_train_model.ipynb)**
  - ➢ Model registration and serving with **MLflow (3-deploy_model.ipynb)**
- **model/config.ipynb**: a config file to adjust the parameters of the project to any specific need.
- **predictor_treatment_outcome.ipynb**: contains a function that serves the deployed model into MLflow. The function is named predict_treatment_outcome and takes a dataframe as parameter. The function returns the prediction of treatment outcomes for the given dataset, appended to the passed dataframe.
- **answers.docx**: the current word file.

Tasks 1

Starting a machine learning project, especially in the context of an adaptive trans-diagnostic intervention program, involves several structured steps. Here's a general outline:

1. **Understanding the domain and Defining the Problem**: The first step is to have a clear understanding of what the problem is and what we are trying to solve. This involves discussions with stakeholders (in this case, management and possibly healthcare professionals) to understand the objectives and desired outcomes of the project, and to understand the domain itself (**Adaptive Trans-diagnostic Intervention program**)
2. **Data Collection and Preparation**: Gather all relevant data. This would include the clinical data (psychological assessments) and behavioral platform data (like login frequency) as mentioned in the problem description. Data preparation includes cleaning, normalizing, and structuring the data for analysis. This step is essential to understand which data do we have in our datastore, and figure out if the existing data is enough to reach our target.
3. **Exploratory Data Analysis (EDA)**: Perform an initial investigation on our data to discover patterns, spot anomalies, test hypothesis, and check assumptions with the help of statistical summaries and graphical representations. This step is essential to conduct fruitful discussion with stakeholders and professionals.
4. **Feature Engineering and Selection**: Identify which data features are most relevant to the problem. This may involve creating new features from the existing data and selecting those that will be most useful for our model.
5. **Choosing the Right Model(s)**: Based on the problem at hand, decide on the appropriate machine learning algorithms.
6. **Training the Model**: Use the prepared dataset to train our model. This involves splitting the data into training, validation and test sets, and possibly using cross-validation techniques to validate the performance of the model.
7. **Model Evaluation and Tuning**: Evaluate the performance of our model using appropriate metrics. Based on the evaluation, we might need to go back and tweak the model parameters, features, or even the model itself.
8. **Deployment**: Once the model is ready and performing well, it's time to deploy it into our production environment where it can start providing insights or predictions. This could be integrating the model into our web/mobile application for real-time interventions.
9. **Monitoring and Maintenance**: After deployment, continuous monitoring is necessary to ensure the model performs as expected over time. The model may require updates or retraining as new data comes in or as the underlying data patterns change.
10. **Feedback Loop**: Implement a mechanism to gather feedback on the model's performance and its impact on the intervention program. We use this feedback for continuous improvement of the model.

The success of a machine learning project **heavily depends on clearly defined objectives, quality of data, and continuous iteration and improvement** based on feedback and changing requirements.

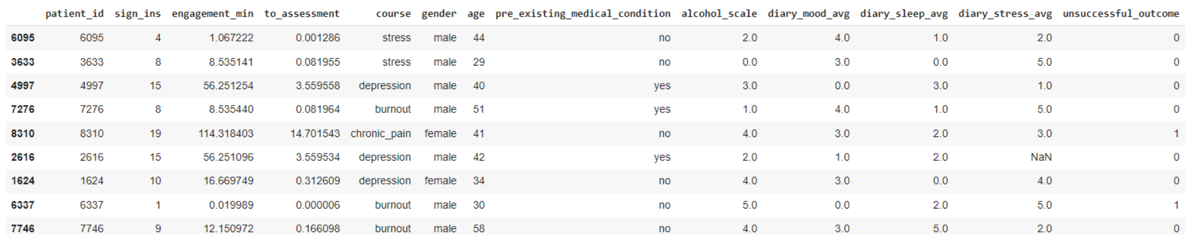
Task 2

1. Understanding the Domain, Problem and Objective

- **Domain**: Adaptive Trans-diagnostic Intervention program, Psychology
- **Objective**: Predict whether a patient will have a positive or negative outcome after completing a specific part of the intervention program.
- **Importance**: Identifying patients who might have a negative outcome allows for timely intervention, improving overall treatment effectiveness.

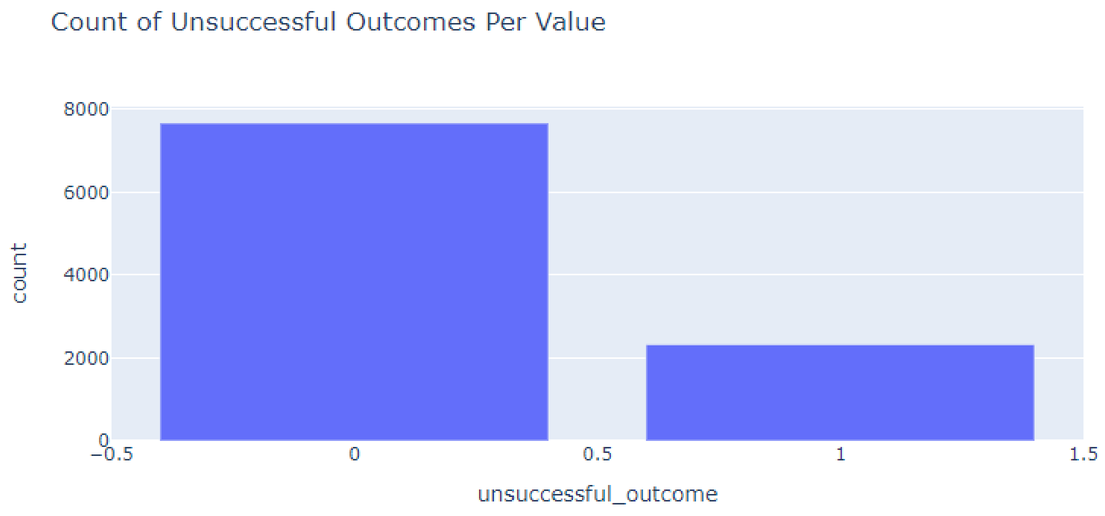
2. Data Exploration and Understanding

The provided dataset has the following **shape (10000, 13).** Below is a sample screenshot of the dataset:

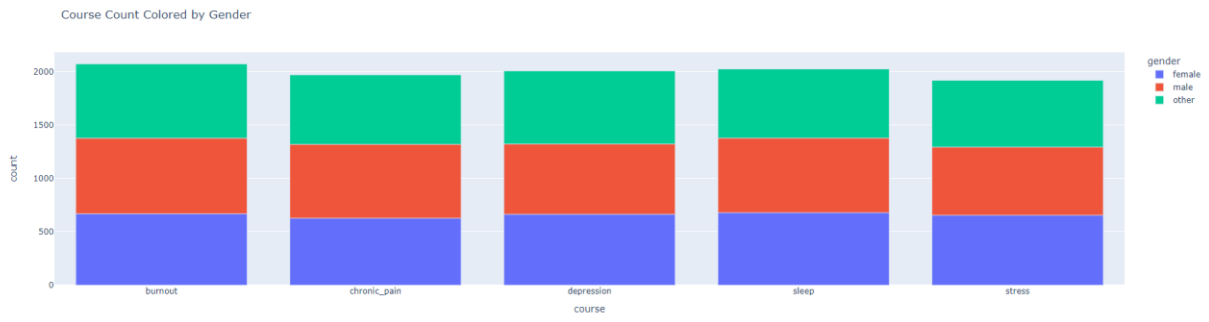


	patient_id	sign_ins	engagement_min	to_assessment	course	gender	age	pre_existing_medical_condition	alcohol_scale	diary_mood_avg	diary_sleep_avg	diary_stress_avg	unsuccessful_outcome
6095	6095	4	1.067222	0.001286	stress	male	44	no	2.0	4.0	1.0	2.0	0
3633	3633	8	8.535141	0.081955	stress	male	29	no	0.0	3.0	0.0	5.0	0
4997	4997	15	56.251254	3.559558	depression	male	40	yes	3.0	0.0	3.0	1.0	0
7276	7276	8	8.535440	0.081964	burnout	male	51	yes	1.0	4.0	1.0	5.0	0
8310	8310	19	114.318403	14.701543	chronic_pain	female	41	no	4.0	3.0	2.0	3.0	1
2616	2616	15	56.251096	3.559534	depression	male	42	yes	2.0	1.0	2.0	NaN	0
1624	1624	10	16.669749	0.312609	depression	female	34	no	4.0	3.0	0.0	4.0	0
6337	6337	1	0.019989	0.000006	burnout	male	30	no	5.0	0.0	2.0	5.0	1
7746	7746	9	12.150972	0.166098	burnout	male	58	no	4.0	3.0	5.0	2.0	0

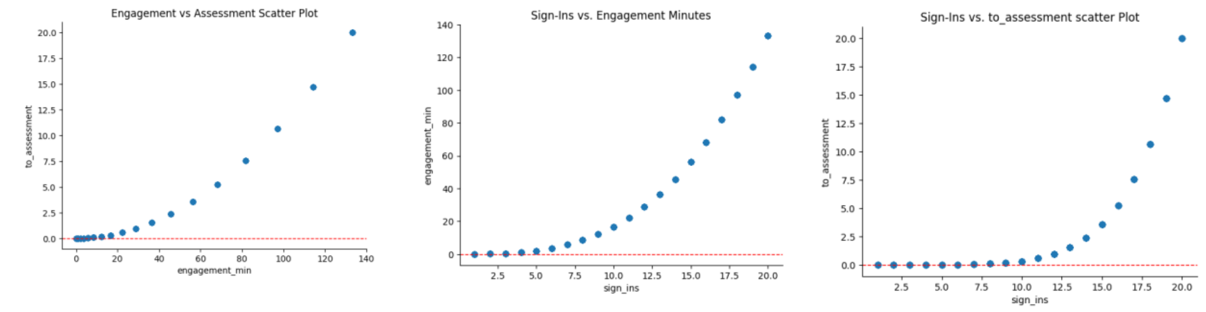
The target column is "**unsuccessful_outcome**". When we analyzed the number of records per each target value, we understood that we have an unbalanced dataset (see screenshot below). Having unbalanced dataset may cause the model to not being able predict under-represented class.



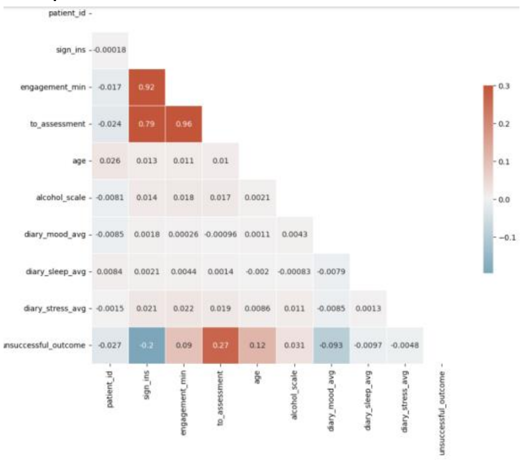
We then tried to check the representation of courses per gender in the dataset, and we got the following plot. We can see that the course/gender ratio is well balanced in the provided dataset:



We also noticed a high correlation between the columns “engagement_min”, “to_assessment” and “sign_ins”:

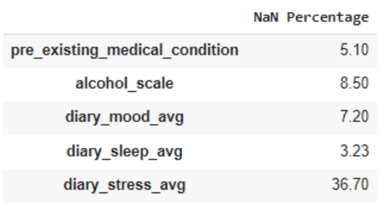


We plotted the correlation matrix between variables:



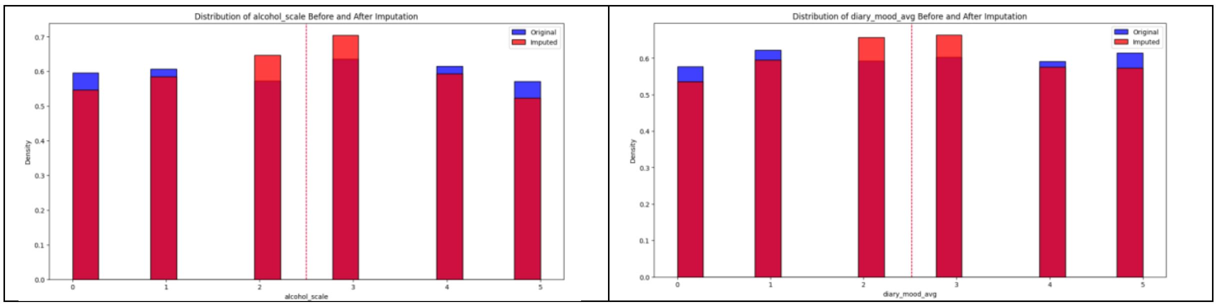
3. Fill NaN values

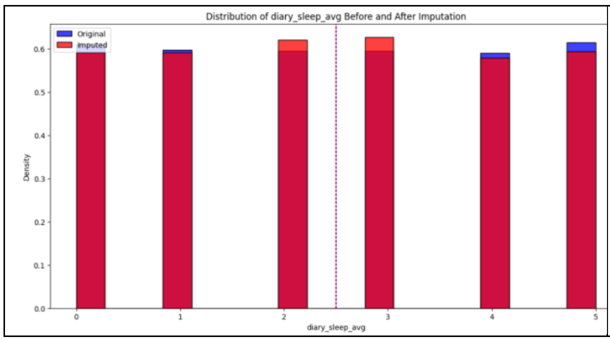
The dataset has NaN values in some columns as described in the table below:

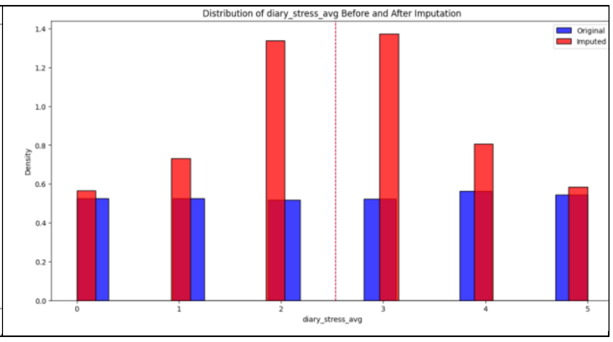


	NaN Percentage
pre_existing_medical_condition	5.10
alcohol_scale	8.50
diary_mood_avg	7.20
diary_sleep_avg	3.23
diary_stress_avg	36.70

To fill them, we trained a **KNN model with k=5**. The plots below describe the distribution of values before and after filling NaN values:







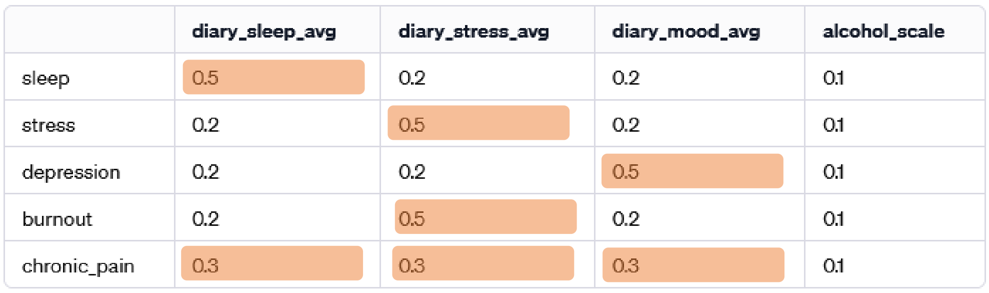
4. Feature Engineering and Selection

After analyzing the features provided by the dataset, we decided to select the following features:

- **Course**
- **Gender**
- **To_assessment :** since the three variables “**engagement_min”,** “**to_assessment**” and “**sign_ins**” are correlated, we will keep only “**to_assessment**” since it provide the same information

We also engineered three other features:

- **combined_to_assessment_age**:
  - we penalize the progress of a user provided by the variable “to_assessment” by multiplying it by 0.01*age. I assumed that age has huge impact on the person’s performance. And thus, even if the person makes progress in the program, age can be an obstacle to transform the progress into success in the program.
- **combined_age_medical_condition**:
  - we assumed that having a medical condition can increase the logical age of the person. We penalize age by multiplying it by 1.5 in case of “pre_existing_medical_condition“.
- **mass_score**:
  - we introduced a new variable that combine the following variables **'alcohol_scale'**, **'diary_mood_avg'**, **'diary_sleep_avg'**, **'diary_stress_avg'.** MASS is the abbreviation of Mood, Alcohol, Stress, Sleep.
  - my research conducted me to understand how adaptive **trans-diagnostic intervention program** works. The idea of this methodology is to not target the disease itself, but its enablers instead.
  - The table below will help to understand how we calculate mass_score. For each treatment program, we assign specific weight to each component:



	diary_sleep_avg	diary_stress_avg	diary_mood_avg	alcohol_scale
sleep	0.5	0.2	0.2	0.1
stress	0.2	0.5	0.2	0.1
depression	0.2	0.2	0.5	0.1
burnout	0.2	0.5	0.2	0.1
chronic_pain	0.3	0.3	0.3	0.1

We ended-up by the following list of features to be provided to the model for training:

- **Course**
- **Gender**
- **Combined_to_assessment_age**
- **Combined_age_medical_condition**
- **Mass_score**

We reduced the number of features from 12 to 5, say **60% reduction of features space**.

5. Feature transformation

I applied log transformation on numerical features.

6. Model training:

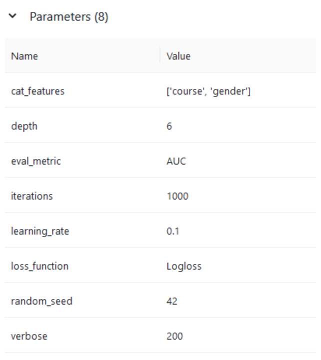
I choose CatBoost model for this purpose. CatBoost handles automatically categorical features (gender and Course) and has a good performance (like most gradient boosting models).

We split the dataset into 3 sub-datasets:

- Training + Validation [cross-validation] (80%)
- Test (20%)

During the training phase, the model never seen the Test dataset.

Below is the list of parameters we set to the model:



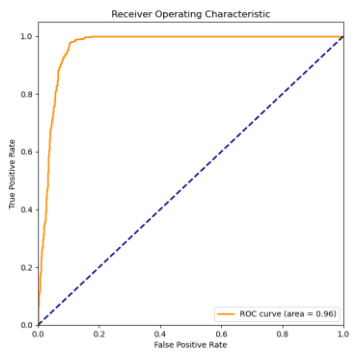
Parameters (8)

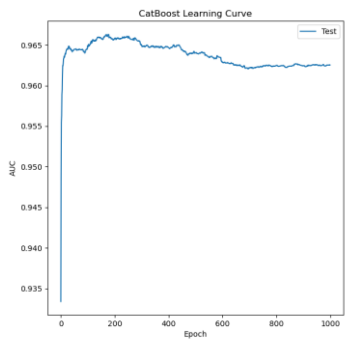
Name	Value
cat_features	['course', 'gender']
depth	6
eval_metric	AUC
iterations	1000
learning_rate	0.1
loss_function	Logloss
random_seed	42
verbose	200

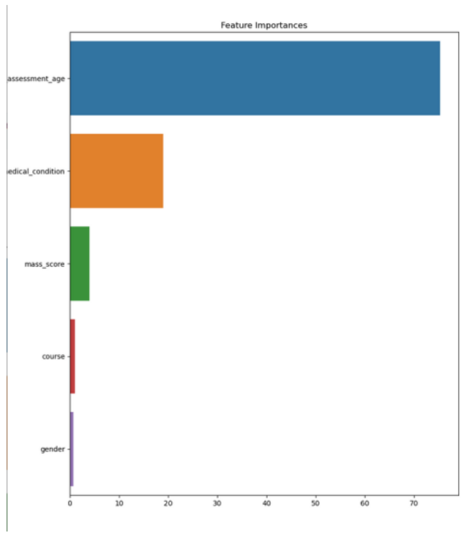
After training we end-up with the following performance on the Test dataset (never seen):

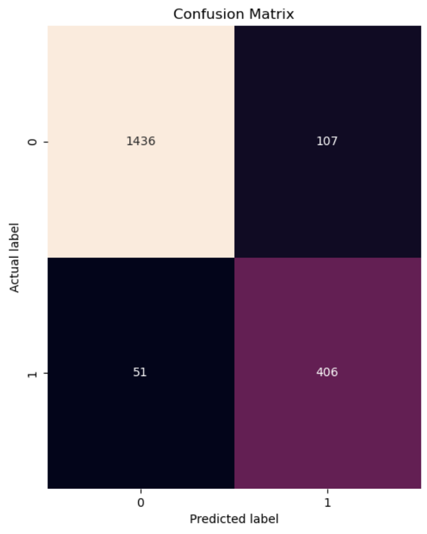
Performance Metrics	Value
ROC-AUC	**96.4 %**
Accuracy	92.1 %

Precision	80.1 %
Recall	88.9 %
F1 score	84 %









7. Model evaluation

Based on the performance plots for the CatBoost model, we can evaluate its performance as follows:

ROC Curve:

- The ROC curve is very close to the top-left corner, which suggests a high true positive rate (TPR) and a low false positive rate (FPR).
- The area under the ROC curve (AUC) is 0.96, which is **quite close to 1, indicating excellent model performance**. A model with perfect discrimination would have an AUC of 1.

Precision-Recall Curve:

- The precision starts high and gradually decreases as recall increases, which is typical as it becomes harder to maintain high precision with a higher recall.

Learning Curve:

- The AUC for the test set is stable after a sharp increase in the initial epochs, hovering around the 0.96 mark.
- There is no significant fluctuation in the AUC value during training, suggesting that the **model has converged well and is stable**.

Confusion Matrix:

- The model has correctly predicted 1436 negative cases (True Negatives) and 406 positive cases (True Positives), which indicates a **strong performance**.
- The number of false negatives (51) and false positives (107) are relatively low compared to the true classifications, further indicating **good performance**.

8. Room for Improvements?

Even with such high scores, we believe we can reach better performance. To do so, we can dedicate more efforts to the below axels:

- Increase dataset size, with more attention to **unsuccessful_outcome ==** 1.
- Revise **mass_score** weights with professionals, along with the other engineered features.
- Write **unit tests** for data transformation pipeline

Task 3

Productionizing a machine learning model is a complex process that involves several steps to ensure the model is **robust, scalable, and maintainable**. Here's a general approach to productionizing the model, along with considerations for deployment and the specific context of using Bitbucket and MLFlow at HelloBetter:

Steps to Productionize the Model:

1. **Version Control**:
   - Use Bitbucket to manage the code base, ensuring all code is version-controlled, including the scripts to train and deploy the model.
   - Keep track of dependencies with a environment.yml.
2. **Testing**:
   - Write unit tests for the code to ensure that every part of the model pipeline is working correctly.
   - Perform integration tests to ensure the entire pipeline from data ingestion to predictions works as expected.
3. **Model Training and Experiment Tracking**:
   - Utilize MLFlow to track experiments, including parameters, metrics, and models.
   - Log the final model along with its version to MLFlow for reproducibility.
4. **Continuous Integration/Continuous Deployment (CI/CD)**:
   - Set up a CI/CD pipeline using tools like Bitbucket Pipelines.
   - Automate testing and deployment processes to ensure reliable, repeatable deployments.
5. **Model Serving**:
   - Decide on a model serving approach: batch, real-time, or streaming. For our case, batch can be a reasonable choice.
   - Use MLFlow Model Serving or integrate with a custom Flask/Django application.
6. **Containerization**:
   - Use Docker to containerize the model and its environment to ensure consistency across development, testing, and production environments.
   - Deploy the container to a container orchestration platform like Kubernetes, which can be managed by cloud services such as AWS EKS or Azure AKS.
7. **Monitoring and Logging**:
   - Implement monitoring for the model's performance and health with tools like Prometheus, Grafana, or cloud solutions (AWS CloudWatch, Azure Monitor).

    - Set up logging to capture predictions, inputs, performance metrics, and errors. This data is crucial for debugging and understanding the model's behavior in production.

8. **Performance Tuning**:
    - Load test the model to understand how it performs under different levels of demand.
    - Tune the infrastructure and model serving parameters to handle the expected load.
9. **Data Management**:
    - Ensure there's a reliable data pipeline for the model, considering data versioning, data quality checks, and easy rollback in case of issues.
    - Use tools like Apache Airflow or Luigi for workflow management.
10. **Security**:
    - Secure the endpoints using authentication, authorization, and encryption.
    - Follow best practices for secure software development and deployment.
11. **Compliance and Ethics**:
    - Consider privacy laws and regulations such as GDPR, HIPAA, etc., and implement necessary compliance checks.

Tools and Technologies:

- **MLFlow**: For experiment tracking, model registry, and deployment.
- **Bitbucket**: For version control and Bitbucket Pipelines for CI/CD.
- **Docker/Kubernetes**: For containerization and orchestration.
- **Prometheus/Grafana**: For monitoring the system's health.
- **Python Libraries**: Such as Flask or FastAPI for creating RESTful services if MLFlow does not suffice.

After deploying the model, continuously monitor its performance and the incoming data distribution. Models in production can drift over time due to changes in the underlying data distribution (concept drift). Set up alerts to notify the team when the model's performance degrades so that we can retrain or update it as necessary.