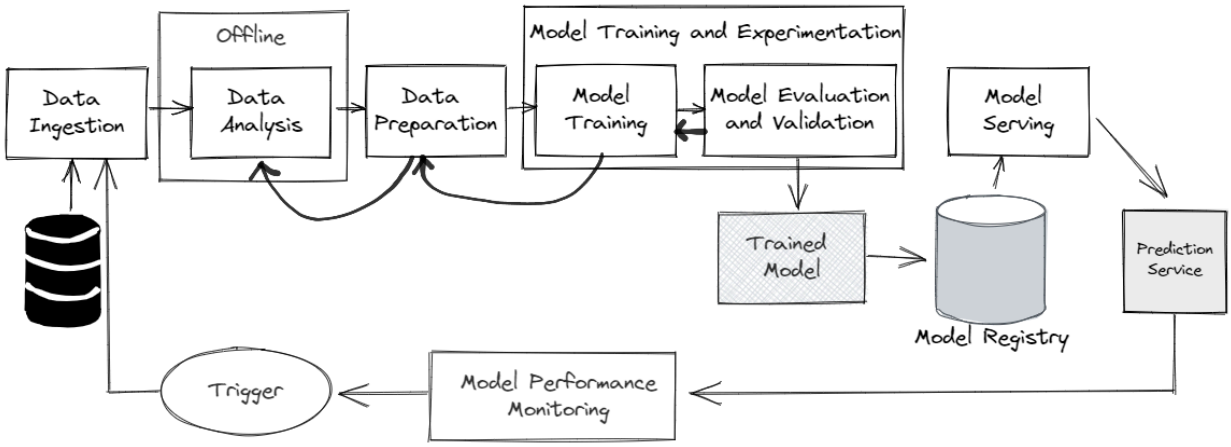
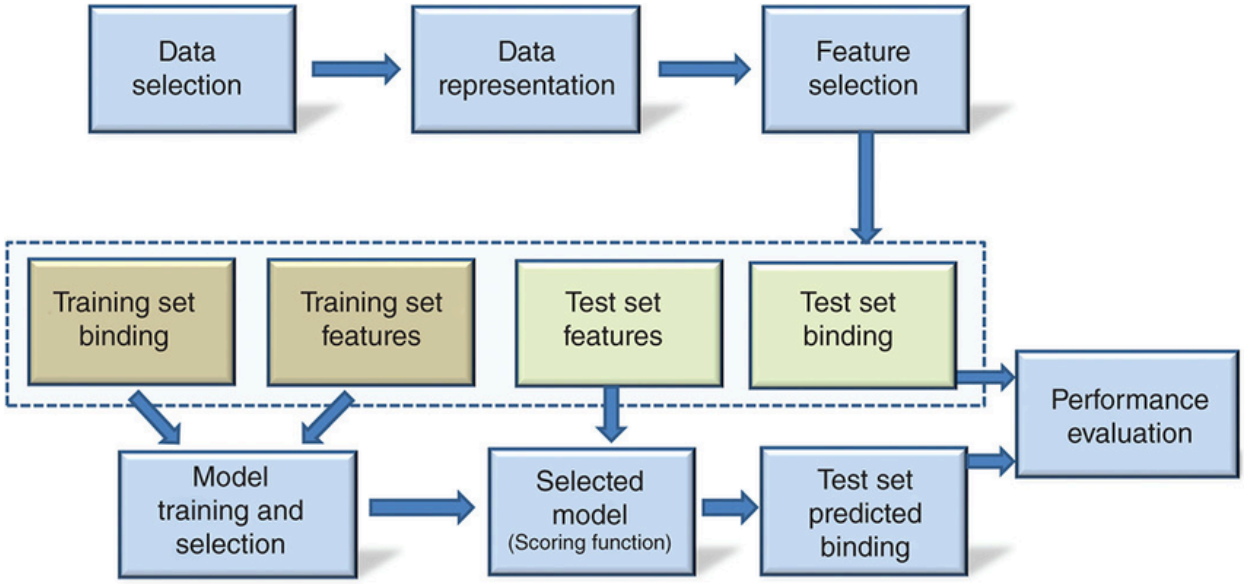
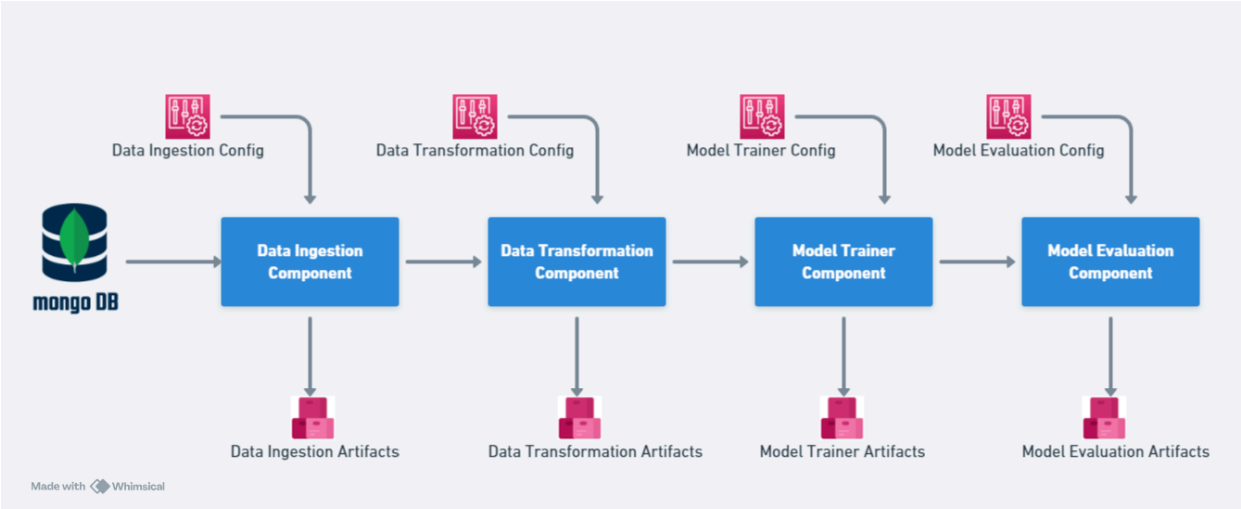
Credit Card Default Prediction

Architecture

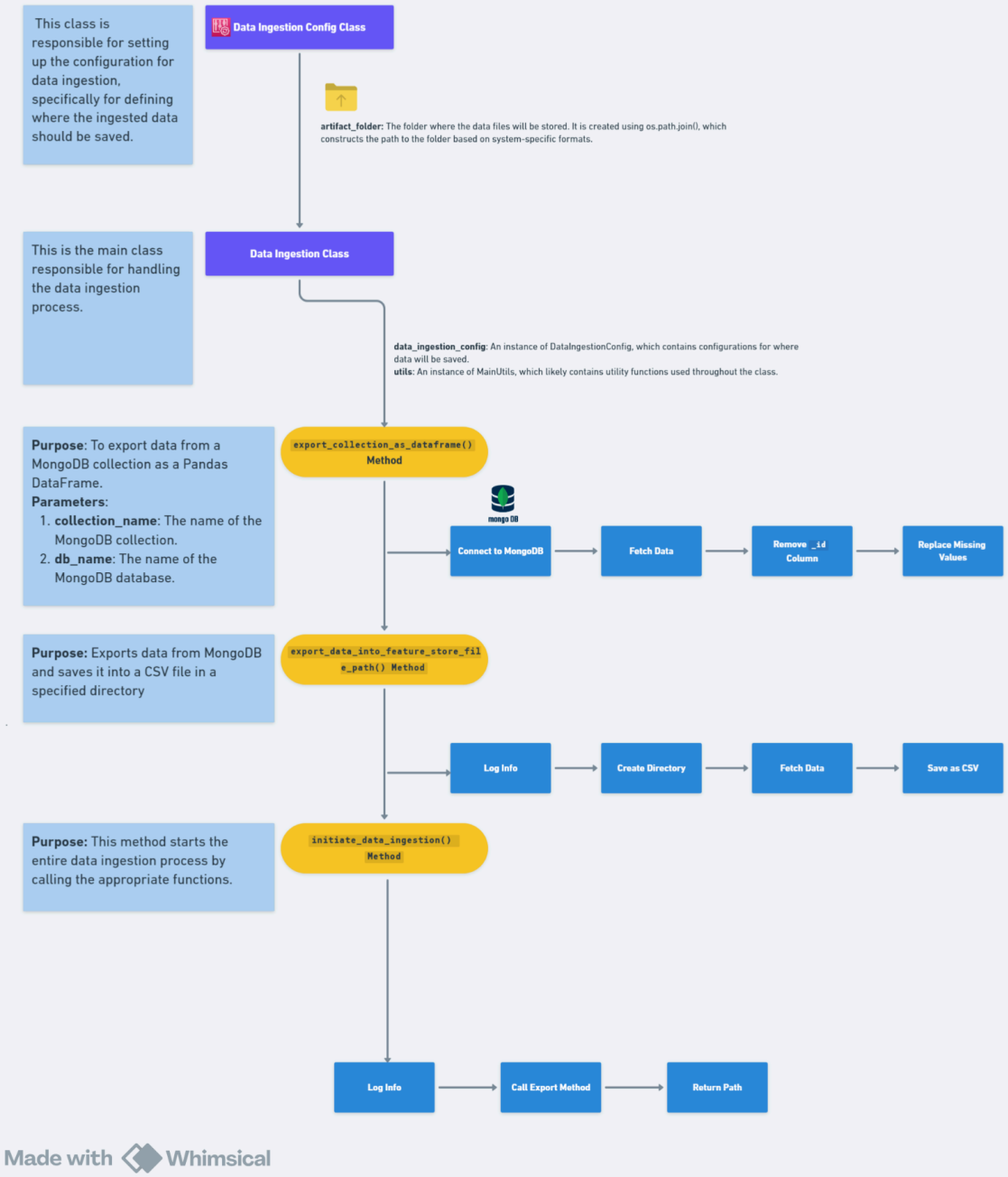
Ritik Patel


Offline
Model Training and Experimentation
Data Ingestion
Data Analysis
Data Preparation
Model Training
Model Evaluation and Validation
Model Serving
Trained Model
Model Registry
Prediction Service
Trigger
Model Performance Monitoring

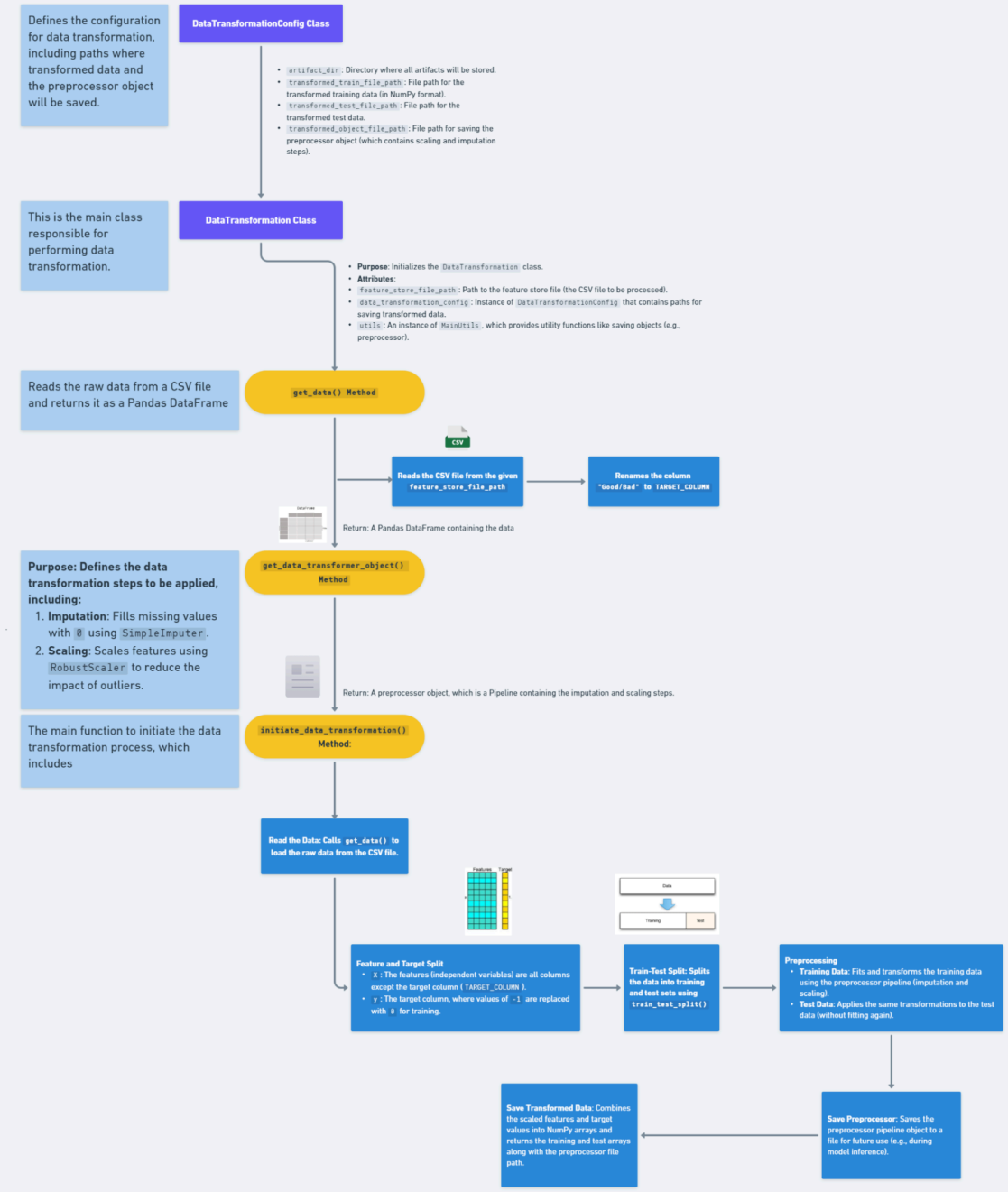

Data selection
Data representation
Feature selection
Training set binding
Training set features
Test set features
Test set binding
Performance evaluation
Model training and selection
Selected model
(Scoring function)
Test set predicted binding


Data Ingestion Config
Data Transformation Config
Model Trainer Config
Model Evaluation Config
mongo DB
Data Ingestion Component
Data Transformation Component
Model Trainer Component
Model Evaluation Component
Data Ingestion Artifacts
Data Transformation Artifacts
Model Trainer Artifacts
Model Evaluation Artifacts
Made with Whimsical

Data Ingestion Code Flow



Data Transformation Code Flow



Model Trainer Code Flow

Purpose: This class defines the configuration for the model training process.

ModelTrainerConfig Class

- `artifact_folder`: Directory where artifacts (like trained models) are stored.
- `trained_model_path`: Path where the final trained model will be saved.
- `expected_accuracy`: The minimum expected accuracy for the model.
- `model_config_file_path`: Path to the configuration file (`model.yaml`) that contains the model hyperparameters.

This is the main class responsible for training, evaluating, and fine-tuning models to find the best one for the given dataset.

ModelTrainer Class

Purpose: Initializes the `ModelTrainer` class.

- **Attributes**:
- `model_trainer_config`: Instance of `ModelTrainerConfig`, which contains the configurations for the model training process.
- `utils`: Instance of `MainUtils`, which provides utility functions like saving objects.
- `models`: Dictionary of machine learning models (XGBoost, Gradient Boosting, SVC, and Random Forest) that will be trained and evaluated.

This method trains and evaluates each model in the `models` dictionary on the training data, then calculates accuracy for both the training and test sets.

`evaluate_models()` Method

Parameters:

- `X`: The feature matrix.
- `y`: The target vector.
- `models`: Dictionary of models to evaluate.

Split the data into training and test sets using `train_test_split()`

For each model in the `models` dictionary:

- Train the model on the training data.
- Predict the labels for both the training and test sets.
- Calculate accuracy for both training and test sets using `accuracy_score()`.
- Store the test accuracy in the `report` dictionary.

Return: A dictionary (report) with model names as keys and their test accuracies as values.

Purpose: Finds the best model based on the accuracy score from `evaluate_models()`

`get_best_model()` Method

Parameters:
`x_train, y_train, x_test, y_test`: Feature and target arrays for training and testing.
Steps:

1. Calls `evaluate_models()` to evaluate the models on the training data.
2. Finds the model with the highest test accuracy from the `model_report`.
3. Returns the best model's name, object, and score.

Purpose: Fine-tunes the best model using `GridSearchCV` to search for the best hyperparameters.

`finetune_best_model()` Method

Parameters:

- `best_model_object`: The best model (selected from `get_best_model()`).
- `best_model_name`: The name of the best model.
- `X_train, y_train`: Training data used for fine-tuning.

Steps:

1. Reads the model's hyperparameter grid from a YAML file using the `MainUtils.read_yaml_file()` method.
2. Performs a grid search on the model to find the best hyperparameters.
3. Updates the best model with the fine-tuned parameters and returns the fine-tuned model.

Purpose: The main method that orchestrates the entire model training, evaluation, and fine-tuning process.

`initiate_model_trainer()` Method

Parameters:

- `train_array, test_array`: Arrays containing training and test data.

Steps:

1. Splits the input arrays into features (`X`) and target (`y`) for both training and testing sets.
2. Calls `evaluate_models()` to get the performance of each model.
3. Identifies the best model using `get_best_model()`.
4. Fine-tunes the best model using `finetune_best_model()`.
5. Trains the fine-tuned model on the training set and evaluates its performance on the test set.
6. If the model meets the accuracy threshold, it saves the model to disk as a `.pkl` file.

Return: The path to the saved model.

Prediction Pipeline Code Flow

Defines the configuration for the prediction pipeline, including paths for models, preprocessors, and prediction outputs.

PredictionPipelineConfig Class

Attributes:
- `prediction_output_dirname` : Directory where the prediction results will be saved.
- `prediction_file_name` : Name of the file where the predictions will be stored.
- `model_file_path` : Path to the serialized machine learning model (`model.pkl`).
- `preprocessor_path` : Path to the preprocessor used for data transformation.
- `prediction_file_path` : Path where the prediction file will be saved.

This is the main class that handles the prediction process.

PredictionPipeline Class

Purpose: Initializes the `PredictionPipeline` class.
Attributes:
- `request` : The incoming request containing the input data (likely an uploaded CSV file).
- `utils` : Instance of `MainUtils` for loading models, preprocessors, and other utility functions.
- `prediction_pipeline_config` : Configuration object containing paths for files and directories.

Saves the input file uploaded by the user into a directory for prediction artifacts.

`save_input_files() Method`

Steps:
1. Create a directory called `prediction_artifacts` if it doesn't exist.
2. Extract the uploaded file from the request and save it to the directory.

Return: Path to the saved input CSV file.

Uses the pre-trained model and preprocessor to make predictions on the input features.

`predict() Method`

Steps:
1. Load the trained model and preprocessor using `MainUtils.load_object()`.
2. Apply the preprocessor to transform the input features.
3. Use the model to predict the output based on the transformed features.

Return: The predicted values.

Reads the input CSV file, makes predictions, and adds a new column for the predictions.

`get_predicted_dataframe() Method`

Steps:
1. Read the input data from the CSV file.
2. Drop any unwanted columns (e.g., `Unnamed: 0`).
3. Call the `predict()` method to get the predictions for the input data.
4. Map the prediction values (`0` or `1`) to human-readable labels (`'bad'` or `'good'`).
5. Save the resulting DataFrame with the predictions to a CSV file.

Output: The CSV file is saved to the specified path, and predictions are logged.

Orchestrates the prediction process by running the entire pipeline.

`run_pipeline() Method`

Steps:
1. Calls `save_input_files()` to save the input file.
2. Calls `get_predicted_dataframe()` to generate predictions and save them to a file.

Return: Returns the configuration object, which contains the file paths used in the prediction process.

Training Pipeline Code Flow