Hi everyone, my name is Robin. I'm happy to present my Machine Learning – AI project. My goal is to ensure that this presentation with a special focus on predictive modeling aligns perfectly with the requirements and responsibilities of this AI Machine Learning Engineer role. Let's dive in!

Slide: **Project Overview & Goals**

- **High False Negative:** Incorrectly NO, **Fraudulent** transactions **incorrectly** flagged as **Non-fraudulent**.
- **High False Positive:** Incorrectly YES, **Non-fraudulent** transactions **incorrectly** flagged as **Fraudulent.**
- **Improved precision and recall** by 8% compare to baseline model i.e. **logistic regression**.

Slide: **Problem Statement – Big Picture**

- **Placement:** Dirty money **integrates** into the financial system. For example, cash deposits are made into bank accounts.
- **Layering:** The money is moved through complex transactions to **offshore** bank. **For instance**: Transferring funds to the bank account of Company **X**, providing a loan to Company **Y**, Company **Y** pays a false invoice to Company **X**.
- **Integrations**: This laundered money is reintroduced into the economy as **legal** funds.

Slide: **Exploratory Data Analysis (EDA)**

- **Data Source:** IBM (Kaggle) – This datasets were created to look like real transactions and come with clear, accurate labels. This paper was first appeared in Neural Information Processing System (NeurIPS) 2023 conference.
- **Data Loading and Inspection:** Data loading source can databases (SQL, NoSQL), APIs, files (CSV, JSON, Parquet), cloud storage (S3, GCS), streaming sources (Kafka, Kinesis), web scraping, IoT sensors, and real-time event streams. Inspect the first few rows to understand and structure and content.
- **Data Shape and Size: Shape**: No of Rows & Columns, **Size**: The dataset occupies Z MB of memory.
- **Data Types**: Identify numerical, categorical, datetime etc. And also need to ensure proper data type of columns conversion like converting strings to datetime. For example: **from_bank** and **to_bank** are Numeric Code of the Bank.
- **Check for Duplicates**: Identify and remove duplicate rows
- **Missing Value**: will take care in data preprocessing stage using pipeline.
- **Data Statistics**: mean, median, **standard deviation** for **numerical columns** and **frequency distribution** for **categorical columns**

Key Features to Explore

- **day**: Banks and financial institutions have fewer staff and lower surveillance, Suspicious activities may go unnoticed until the next business day, giving launderers more time to hide their actions.
- **timestamp**: Check if fraud occurs more frequently at specific times (e.g., hours, days).
- **payment_format**: ACH (**Automated Clearing House**) handle **many transactions**, making it hard to spot illegal ones, let money move **quickly and easily**, hiding where it came from. Also not closely watched in real-time.
- **receiving_currency and payment_currency**: We need to check if certain currencies are more prone to fraud.

- **from_bank and to_bank**: Analyze if specific banks are more involved in fraudulent transactions.
- **amount_received and amount_paid**: Both contains significant outliers, distribution is highly right-skewed, few transactions extremely high values, they have similar distributions. **Solution**: The presence of extreme values suggests the need for **log transformation** or **robust scaling** to improve model performance. **Robust Scaling** transforms the data using **the median and interquartile range (IQR)**.

Slide: **Feature Engineering**

It includes techniques like creating new features, encoding categorical variables, and scaling numerical values to enhance predictive accuracy.

- **Time-Based Features**: Extract hour, day, or month from timestamp.
- **Amount Discrepancy**: Difference between amount_received and amount_paid.
- **Currency Mismatch**: Flag transactions where receiving_currency != payment_currency.
- **Transaction Frequency**: Number of transactions per account or bank.
- Let's say we have column with Text**,** use **BERT**
- Using **PCA** reduced to reduce to dimension

Feature Selection: Filter Method (chi-square test**), Wrapper Method (***Recursive Feature Elimination (RFE)***), Embedded Methods (**lasso, ridge, tree-based**), Dimensionality Reduction (**PCA**), Model-Based (**using ML models to rank**), Domain Knowledge**

Numerical Features:

- VIF (**variance inflation factor**) is used to assess the multicollinearity among the independent (predictor) variables
- VIF measures how much the variance of a coefficient is inflated due to correlations with other predictors
- Most regression models include an intercept term (**constant**) to account for the baseline value of the dependent variable when all independent variables are zero.

Categorical Features:

- A chi-square statistic is one way to show a relationship between two categorical variables. We test correlation of **Categorical** columns with **Target** column i.e *is_laundering* assess the multicollinearity among the independent (predictor) variables
  - **p-value** is the smallest level of significance at which we Reject the Hypothesis. It measures the strength of evidence against a null hypothesis

```
chi2_test = []
for feature in categorical_features:
    if chi2_contingency(pd.crosstab(df['is_laundering'], df[feature]))[1] < 0.05:
        chi2_test.append('Reject Null Hypothesis - There is a relationship')
    else:
        chi2_test.append('Fail to Reject Null Hypothesis - There is no relationship')
result = pd.DataFrame(data=[categorical_features, chi2_test]).T
result.columns = ['Column', 'Hypothesis Result']

# Increase the display width of the "Hypothesis Result" column
```

- `pd.set_option('display.max_colwidth', 100)  # Set the maximum column width to 100 characters`
-
- The "Reject Null Hypothesis - There is a relationship" result suggests that these variables have a statistically significant relationship with the target variable being analyzed. This means that these variables are likely to be useful predictors in a model.

- Conversely, the variables with "Fail to Reject Null Hypothesis - There is no relationship" are less likely to be helpful as predictors. While they might have some indirect influence or interact with other variables, their direct predictive power appears to be weak based on this analysis. Including them might add noise to the model and decrease its performance.

Why Multicollinearity Doesn't "Matter" as Much in RF/XGBoost:

- **Tree-based models select variables:** They inherently choose the most important variables at each split. Multicollinearity means correlated features carry similar information. If one correlated feature is selected, its "redundant" partner is less likely to be used further down the tree.

Why Removing Highly Correlated Variables Can Still Be Good:

- **Model Simplicity/Interpretability:** Fewer features make the model easier to understand.
- **Computational Efficiency:** Less data to process leads to faster training and prediction.
- **Potential Overfitting (rare):** If two features are *almost* perfectly correlated and add noise, keeping only one can reduce overfitting

Slide: **Data Preprocessing**

Slide: **Model Development and Evaluation**

Model Selection and Training:

- **Model Selection**: Random Forest, AdaBoost, XGBoost → AML data is rarely linear. These models can capture complex, **non-linear** relationships between features (transaction amounts, sender/receiver history, locations, etc.) to identify patterns indicative of fraud.
- **Base Model**
  Logistic Regression often underperforms compared to **Random Forest** and **XGBoost** for **AML Fraud Detection** due to: Fraud patterns are often complex and **non-linear**, Random Forest and XGBoost automatically handle feature interactions, while Logistic Regression requires manual feature engineering. Tree-based models (Random Forest, XGBoost) are less sensitive to outliers compared to Logistic Regression.

- **Hyperparameters T**unning (GridSearch) →

Random Forest:

- `n_estimators`: (Number of trees) - *Impacts accuracy, tune for diminishing returns.*
- `max_depth`: (Tree depth) - *Controls complexity, prevent overfitting, tune by observing the performace.*
- `class_weight`: (Class balance) - *Crucial for imbalanced data ("balanced").*

AdaBoost:

- `n_estimators`: (Number of learners) - *Similar to Random Forest, tune for diminishing returns.*

- **`learning_rate`:** (Step size) - *Prevents overfitting, smaller values need more estimators.*

XGBoost:

- **`n_estimators`:** (Boosting rounds) - *Tune with early stopping.*
- **`learning_rate`:** (Step size) - *Prevents overfitting, smaller values need more rounds.*
- **`max_depth`:** (Tree depth) - *Control complexity, helps to avoid overfitting.*
- **`scale_pos_weight`:** (Class balance) - *Essential for imbalanced data; balance the class weights.*
- **`subsample`:** (Subsample ratio) - *Reduces variance.*
- **`colsample_bytree`:** (Feature subsample) - *Reduces correlation between trees.*

- **Cross-Validation:** Use k-fold cross-validation to evaluate the model's performance on unseen data and avoid overfitting.
- **Grid Search/Randomized Search:** Use grid search or randomized search to systematically explore the hyperparameter space.
- **Early Stopping:** Use early stopping (especially with XGBoost) to stop training when the model's performance on a validation set starts to degrade. This prevents overfitting and saves training time.

Random Forest **doesn't** have a traditional **optimizer** like gradient descent; it builds independent, randomized trees using a greedy algorithm for feature selection and split point selection. "Optimization" comes from averaging the predictions of these diverse trees, and hyperparameter tuning is done externally to configure the algorithm.
XGBoost ***does* have an optimizer**: it uses gradient boosting, which iteratively adds trees, each attempting to correct errors from previous ones by minimizing a loss function via gradient descent, hence the optimizer.

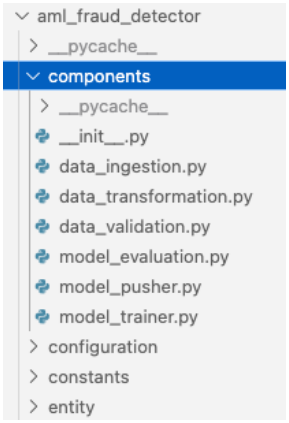
Performance Metrics:

- Precision (Reduce False Positive), Recall (Reduce False Negative)

Slide: **Deployment with GitHub Actions and AWS**

Model Deployment:

- **setup.py**: will be responsible in creating “machine learning” application as a package. And deploy in
- src (**aml_fraud_detector**) – we need to create a file “__init__.py” inside the “aml_fraud_detector” to found as a package. Also it can be imported to some other file location in project folder.



- **data_ingestion.py** -> All codes relative to "reading" the data... source can databases (SQL, NoSQL), APIs, files (CSV, JSON, Parquet), cloud storage (S3, GCS), streaming sources (Kafka, Kinesis), web scraping, IoT sensors, and real-time event streams.

- **data_validation.py** -> We need to data validation, for example, let say new incoming data might be one of the columns (or features). Other case will be, let's say you train your model with training data, and the unseen data (or test data) missing some features or having more features. **How to resolve it**? Identify the missing features and their importance -> If unimportant, retrain the model without them ->If important, impute or engineer replacements -> Monitor the system to prevent future occurrences.

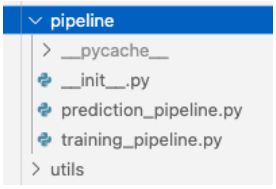
Data Drift: The **test (or production) data** change over time compared to the **training data** it was originally trained on. I have used this using **Evidently AI** (Evaluate, Test, Monitor). Track data changes, update the model with new data, **fix unstable features**, **adapt to new patterns**, use real-time feedback, **add more relevant data**, and make the model more resilient to changes

- **data_transformation.py** -> removing duplicates, creating features, feature engineering, data preprocessing with make_pipeline (column transform for numeric and categorical)

- **model_trainer.py** -> different kind of model, their hyperparameters, metrics information

- **model_evaluation.py** -> we MLflow

- **model_pusher.py** -> After model evaluation and the model performance meet the accuracy or precision threshold like 80%, and if the production model is less than 80% then new model will be pushed. Then the model will pushed to **AWS S3** for example



- **training_pipeline.py** ->

- **prediction_pipeline.py** -> where I create code for streamlit application, where I will load pickle files "best model" object, "data_preprocessor" object. And when the user input the information web interface, that user input data will preprocess and then the trained best model will make prediction on those input data.

- Now, run the "app_streamlit.py" using "**`streamlit run app_streamlit.py`**"

Cloud Deployment:

- **AWS Elastic Beanstalk** : is a service that helps you deploy and scale web applications on Amazon Web Services (AWS). .ebextensions ->python.config -- Elastic Beanstalk is primarily designed for **WSGI-based applications** (like Flask or Django), and Streamlit is not natively supported as a WSGI app. Where we can deploy connecting "Github" using "AWS Code Pipeline" with "AWS **Elastic Beanstalk"**
- **Docker:**
  - **==> `docker images` -> To check docker images**
  - **==> `docker build -t aml-streamlit-app .` ->To build docker images**
  - **==> `docker run -p 8501:8501 aml-streamlit-app`**

==> `docker ps -a` -> to check app running in a container

==> `docker stop <container ID>` ex:50d3d2bba4ad

==> `docker rmi <image_id_or_name> or docker rmi -f aml_streamlit-app`

==> `docker image prune` -> Remove all unused images

==> `docker tag old_dockerImage_name new_dockerImage_name` -> Change docker image name

==> `docker push welcome-app:latest` -> Push docker image to "DockerHub"

sudo lsof -i :8501 → to check port

kill -9 <PID> → to stop process

1. **CI/CD Setup with GitHub Actions:**
   - Configured GitHub Actions workflows in the `.github/workflows` directory, the `main.yaml` file, to automate the build, test, and deployment stages.

Workflow Summary for **main.yaml** file

(Continuous Integration) Integration: Checks out the code, lints it, and runs unit tests on every push to the main branch (excluding changes to README.md).

(Continuous Delivery): build-and-push-ecr-image Triggered after successful integration. It checks out the code, configures AWS credentials, logs into Amazon ECR, builds a Docker image, tags it as latest, and pushes it to the specified ECR repository.

(Continuous-Deployment): Triggered after the image is successfully pushed to ECR. Deploys the updated container to a self-hosted environment. It authenticates with AWS, pulls the latest image, and then runs the docker with the latest image. Also it clean the previous container and old images to save resources.

2. **AWS Console Setup:**
   - **Login to AWS Console:** Accessed the AWS Management Console to set up required resources.
   - **IAM User Creation:** Created an IAM user with permissions for deployment:
     - `AmazonEC2ContainerRegistryFullAccess`: Full access to Amazon Elastic Container Registry (ECR).
     - `AmazonEC2FullAccess`: Full access to Amazon EC2.
3. **Elastic Container Registry (ECR) Setup:**
   - Created an ECR repository for storing Docker images.
   - **ECR Repo URI:** `767397970670.dkr.ecr.us-east-1.amazonaws.com/aml_fraud_detector-container`
4. **EC2 Instance Setup:**
   - Create EC2 (Ubuntu): Virtual machine in the AWS cloud
   - Launched an EC2 instance (Ubuntu) to run the application.

```
# optional
sudo apt-get update -y
sudo apt-get upgrade

# Required Need to install docker in EC2 instance just created
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo usermod -aG docker ubuntu
newgrp docker
```

   - **Connect to EC2 Instance:** Built Docker image, pushed it to ECR, and launched it on EC2.

        - **Description**: About the deployment
    b. Build docker image of the source code
    c. Push docker image to ECR
    d. Launch EC2
    e. Pull image from ECR in EC2
    f. Launch docker image in EC2
        - Docker setup in EC2
5. Configure EC2 as self-hosted runner in GitHub
    - Now, Go to GitHub

```
setting > actions > runner > new self hosted runner > choose os (Linux) > then run command one by one
```

6. Setup github secrets:

```
setting > Secrets and variables > actions > New repository secret (in main screen)

        AWS_ACCESS_KEY_ID=
        AWS_SECRET_ACCESS_KEY=
        AWS_REGION = us-east-1
        AWS_ECR_LOGIN_URI = 767397970670.dkr.ecr.us-east-1.amazonaws.com
        ECR_REPOSITORY_NAME = aml_fraud_detector-container
```

Slide: **User Interface (Streamlit)**

- To make sure the deployed AI/ML model provides practical insights and actionable results, I developed a user-friendly UI using Streamlit.

Slide: **Business Impact**

- **Reduced Financial Losses:** By improving the detection of fraudulent transactions (reducing False Negatives), the project helps prevent financial losses caused by money laundering activities.
- **Enhanced Customer Trust:** By reducing legal transactions flagged as laundery, the project minimizes disruptions to genuine customers.
- **Improved Operational Efficiency:** Automating fraud detection using machine learning models (e.g., Random Forest, XGBoost) reduces the need for manual review of transactions.
- The project ensures compliance with AML regulations by accurately identifying and reporting suspicious activities. Enhances the organization's reputation as a trustworthy