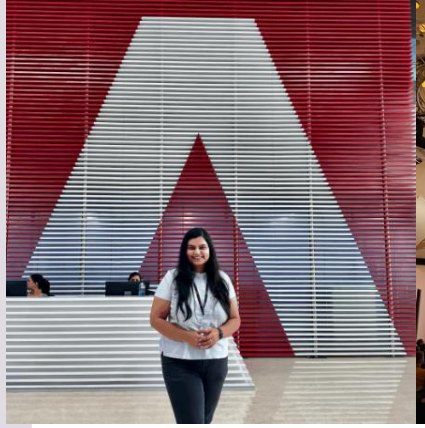
REAL-TIME PREDICTIONS WITH POSTGRESQL AND PL/PYTHON

Drishti Jain

@drishtijjain





About Me

- Software Developer / ML
- Published Technical Book
- Social Entrepreneur
- International Tech Speaker

- Career Coach – SkillUp with Drishti

@drishtijjain

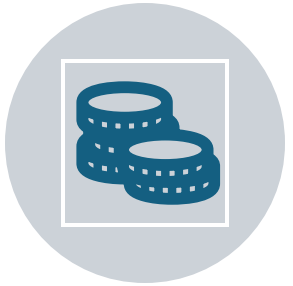

Why Real-Time Predictions?

Real-Time Predications

Instant decision-making

Improved user experience

Competitive advantage



Cost-effective solution

PostgreSQL Features

ROBUST RELATIONAL DATABASE

EXTENSIBILITY WITH PROCEDURAL LANGUAGES

BUILT-IN SUPPORT FOR JSON AND OTHER DATA TYPES

@drishtijjain

PL/Python Advantages

Write PostgreSQL functions in Python

Access to Python's rich ecosystem of libraries

Seamless integration with database operations

Implementing ML Models in PL/Python

@drishtijjain

Implementing ML Models in PL/Python

The ability to implement ML models directly within the database using PL/Python opens new possibilities for real-time predictions and data-driven decision making.

@drishtijjain

Serialization and Deserialization of ML Models

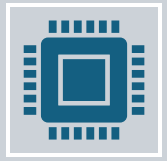
- Serialization is the process of converting a complex object, like a trained machine learning model, into a format that can be easily stored or transmitted.

- Deserialization is the reverse process, reconstructing the object from the serialized format.

@drishtijjain

Database storage

Databases typically store structured data, not complex Python objects.



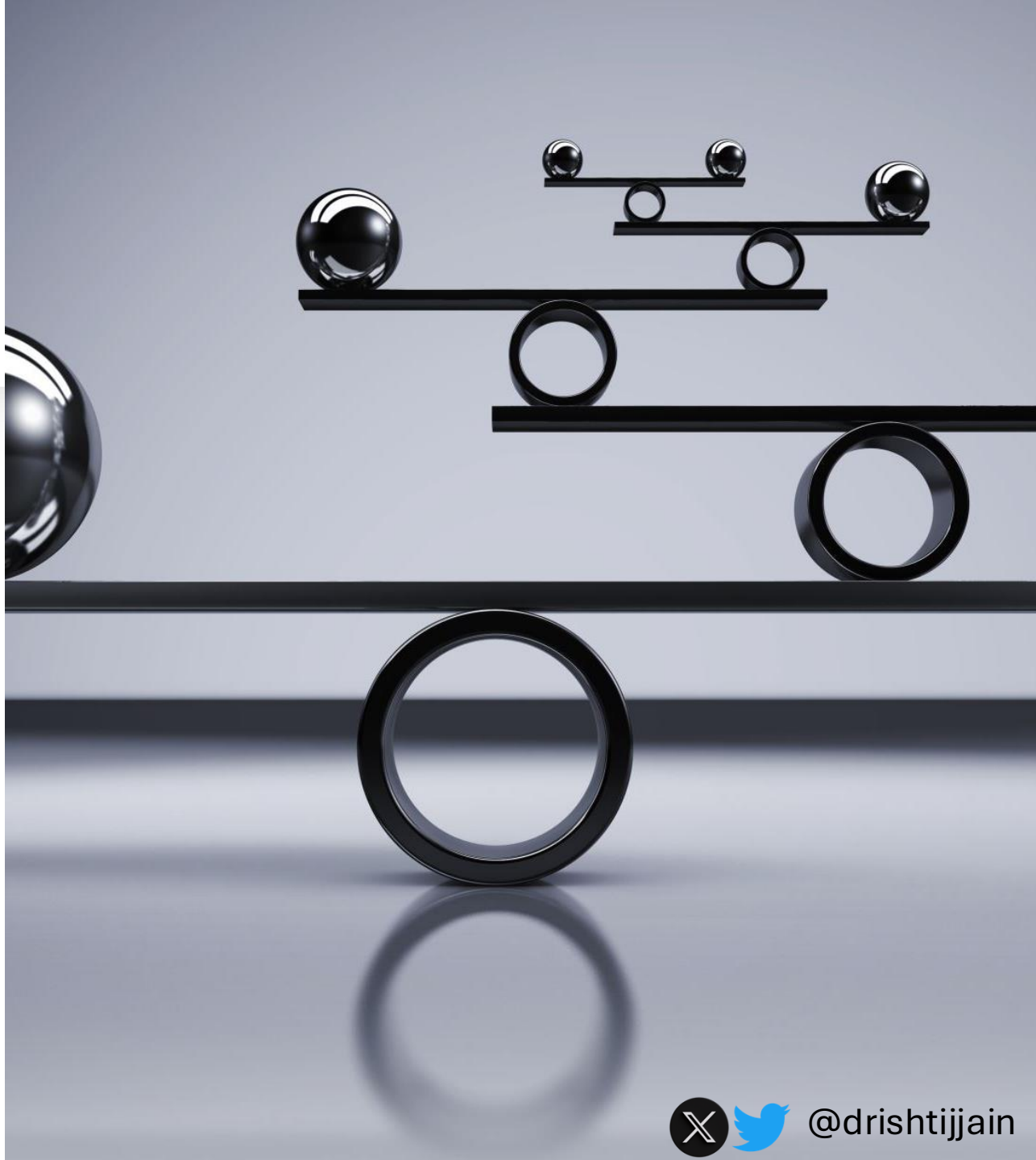
Serialization allows us to convert our ML models into a format (like binary data) that can be stored in database columns.

Data transfer

When moving models between systems or from the training environment to the production database, serialization enables easy transfer of the model.

Persistence

Serialization allows us to save the state of a trained model, so we can load it later without retraining.



In our PostgreSQL and PL/Python context, we're using Python's pickle module for serialization.

Here's how it works:

After training the model, we use pickle.dumps() to serialize it into a binary format.

This binary data is stored in a PostgreSQL bytea column.

When we need to use the model, we retrieve the binary data and use pickle.loads() to deserialize it back into a Python object.

Setting Up the Environment

Enabling PL/Python

```
CREATE EXTENSION plpython3u;
```

Creating a Sample Database

```
CREATE TABLE customer_data (
    id SERIAL PRIMARY KEY,
    age INTEGER,
    income NUMERIC,
    credit_score INTEGER,
    purchase_amount NUMERIC
);

INSERT INTO customer_data (age, income, credit_score, purchase_amount)
VALUES
    (35, 50000, 700, 1000),
    (28, 35000, 650, 500),
    (45, 80000, 750, 2000);
```

Implementing ML Models in PL/Python

@drishtijjain

Linear Regression Model

@drishtijjain

Linear regression

- Linear regression is a statistical method for modeling the relationship between a dependent variable (Y) and one or more independent variables (X)
- It assumes a linear relationship between variables
- The simplest form is simple linear regression with one independent variable:

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

Where:

- β_0 is the y-intercept
- β_1 is the slope
- ε is the error term

@drishtijjain

Linear Regression Model

```
CREATE OR REPLACE FUNCTION train_model()
RETURNS bytea AS $$
    import pandas as pd
    from sklearn.linear_model import LinearRegression
    import pickle

    # Fetch data from the database
    data = plpy.execute("SELECT age, income, credit_score, purchase_amount FROM customer_data")
    df = pd.DataFrame(data)

    # Prepare features and target
    X = df[['age', 'income', 'credit_score']]
    y = df['purchase_amount']

    # Train the model
    model = LinearRegression()
    model.fit(X, y)

    # Serialize the model
    return pickle.dumps(model)
$$
 LANGUAGE plpython3u;
```

@drishtijjain

Linear Regression Model

```
-- Train and store the model
CREATE TABLE IF NOT EXISTS ml_models (
    id SERIAL PRIMARY KEY,
    model_name TEXT,
    model_data bytea
);

INSERT INTO ml_models (model_name, model_data)
VALUES ('purchase_predictor', train_model());
```

@drishtijjain

Creating a Prediction Function

Prediction Function

```
CREATE OR REPLACE FUNCTION predict_purchase(age INTEGER, income NUMERIC, credit_score INTEGER)
RETURNS NUMERIC AS $$
    import pickle
    from sklearn.linear_model import LinearRegression

    # Fetch the model
    model_data = plpy.execute("SELECT model_data FROM ml_models WHERE model_name = 'purchase_predictor'")[0]['model_data']
    model = pickle.loads(model_data)

    # Make prediction
    prediction = model.predict([[age, income, credit_score]])
    return prediction[0]
$$
 LANGUAGE plpython3u;
```

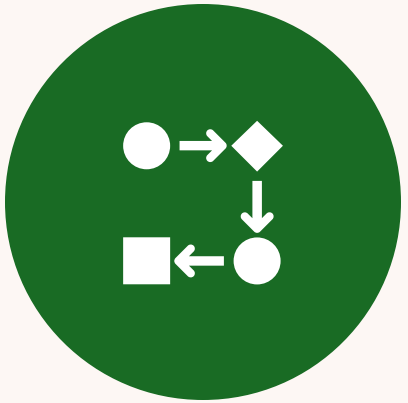
@drishtijjain

Trade-offs in a Database Context

@drishtijjain

Model Complexity vs. Prediction Speed

MORE COMPLEX MODELS (LIKE DEEP NEURAL NETWORKS) MAY PROVIDE HIGHER ACCURACY BUT SLOWER PREDICTION TIMES.



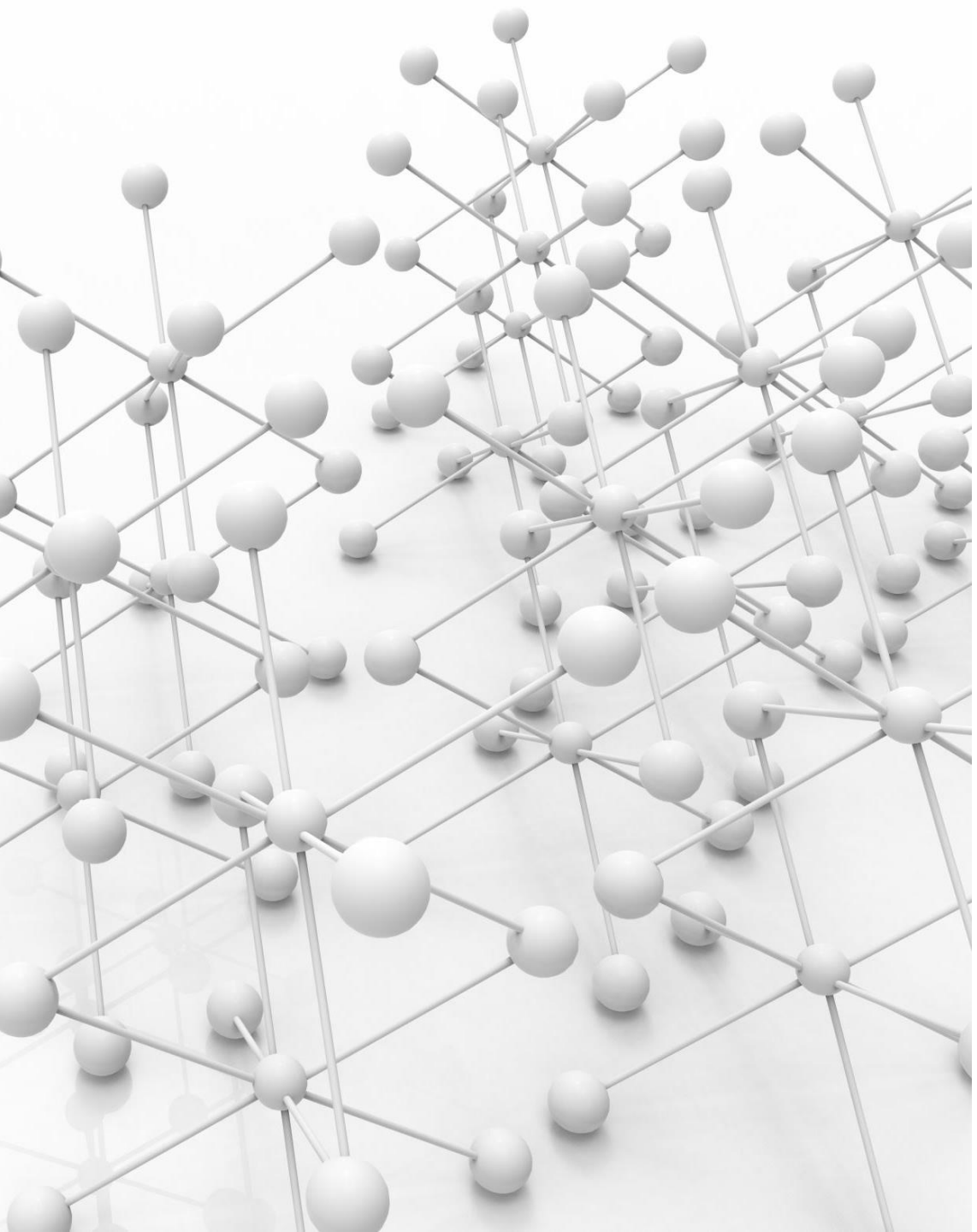
SIMPLER MODELS (LIKE LINEAR REGRESSION OR SMALL DECISION TREES) OFFER FASTER PREDICTIONS BUT MAY SACRIFICE SOME ACCURACY.

IN A DATABASE CONTEXT WHERE REAL-TIME PREDICTIONS ARE OFTEN NEEDED, YOU MAY NEED TO BALANCE ACCURACY WITH SPEED.

@drishtijjain

Memory Usage:

1. More complex models require more memory, which could impact database performance.
2. Consider the scalability of your solution as the number of concurrent predictions increases.



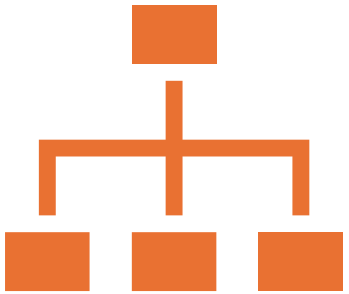
Interpretability

In many business contexts, being able to explain predictions is crucial.

While a complex neural network might provide high accuracy, a simpler model like a decision tree might be preferred for its interpretability.

@drishtijjain

Training and Updating



Consider how often the model needs to be retrained or updated.

More frequent updates might favor simpler models that can be quickly retrained within the database.

@drishtijjain

Data Volume and Velocity

If you're dealing with high-volume, high-velocity data, you might need to opt for models that can handle streaming data effectively.

Regulatory Compliance

In some industries, regulatory requirements might limit the types of models you can use, favoring more interpretable models.

Real-Time Prediction Pipeline

@drishtijjain

Integrating Predictions into Queries

```
SELECT
    id,
    age,
    income,
    credit_score,
    predict_purchase(age, income, credit_score) AS predicted_purchase
FROM customer_data;
```

@drishtijjain

Trigger

Triggers for automating predictions

FRAUD DETECTION IN FINANCIAL TRANSACTIONS

REAL-TIME PRICING IN E-COMMERCE

PREDICTIVE MAINTENANCE IN INDUSTRIAL SETTINGS

PERSONALIZED CONTENT RECOMMENDATIONS

@drishtijjain

Triggers for automating predictions

Challenges:

- Database Performance
- Scalability concerns
- Keeping models up-to-date

@drishtijjain

```
CREATE OR REPLACE FUNCTION update_prediction()
RETURNS TRIGGER AS $$
BEGIN
    NEW.predicted_purchase = predict_purchase(NEW.age, NEW.income, NEW.credit_score);
    RETURN NEW;
END;
$$
 LANGUAGE plpgsql;

CREATE TRIGGER predict_on_insert
BEFORE INSERT ON customer_data
FOR EACH ROW
EXECUTE FUNCTION update_prediction();
```

Creating a Trigger for Real-Time Predictions

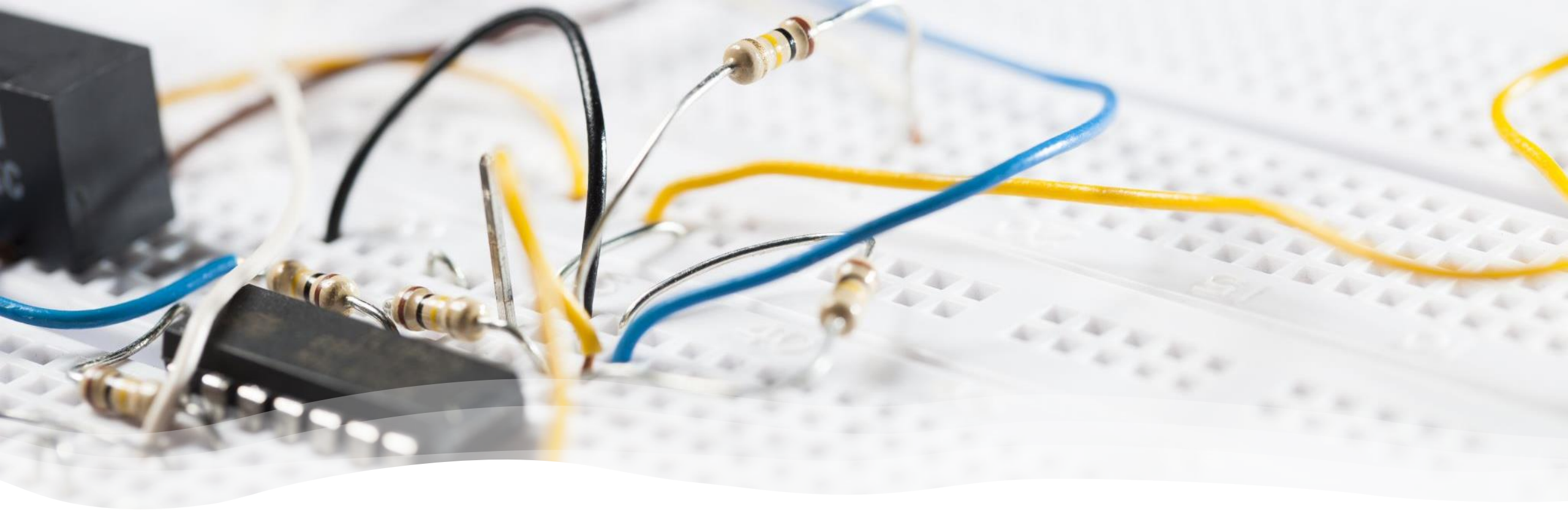
- We can automate the prediction process using PostgreSQL triggers
- This trigger will automatically calculate and store a prediction when new data is inserted

Inserting New Data and Getting Real-Time Predictions

```
INSERT INTO customer_data (age, income, credit_score)
VALUES (30, 60000, 720);

SELECT * FROM customer_data ORDER BY id DESC LIMIT 1;
```

@drishtijjain



Retrain the Model with all available data (NEW + Exisitng)

@drishtijjain

Updating the Model

```
-- Update the model with new data
UPDATE ml_models
SET model_data = train_model()
WHERE model_name = 'purchase_predictor';
```

@drishtijjain

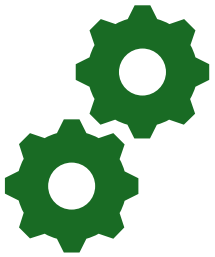
Best Practices

Best Practices

1. Regularly update and retrain models
2. Monitor model performance
3. Handle errors gracefully
4. Optimize for performance

Takeaways

PostgreSQL with PL/Python enables powerful real-time predictions



Seamless integration of ML models into database operations

Potential for enhanced decision-making and user experiences

@drishtijjain

THANK YOU!

@drishtijjain

linkedin.com/in/jaindrishti/

medium.com/@drishtijjain

@geekyearthian