MIS710 – Machine Learning in Business

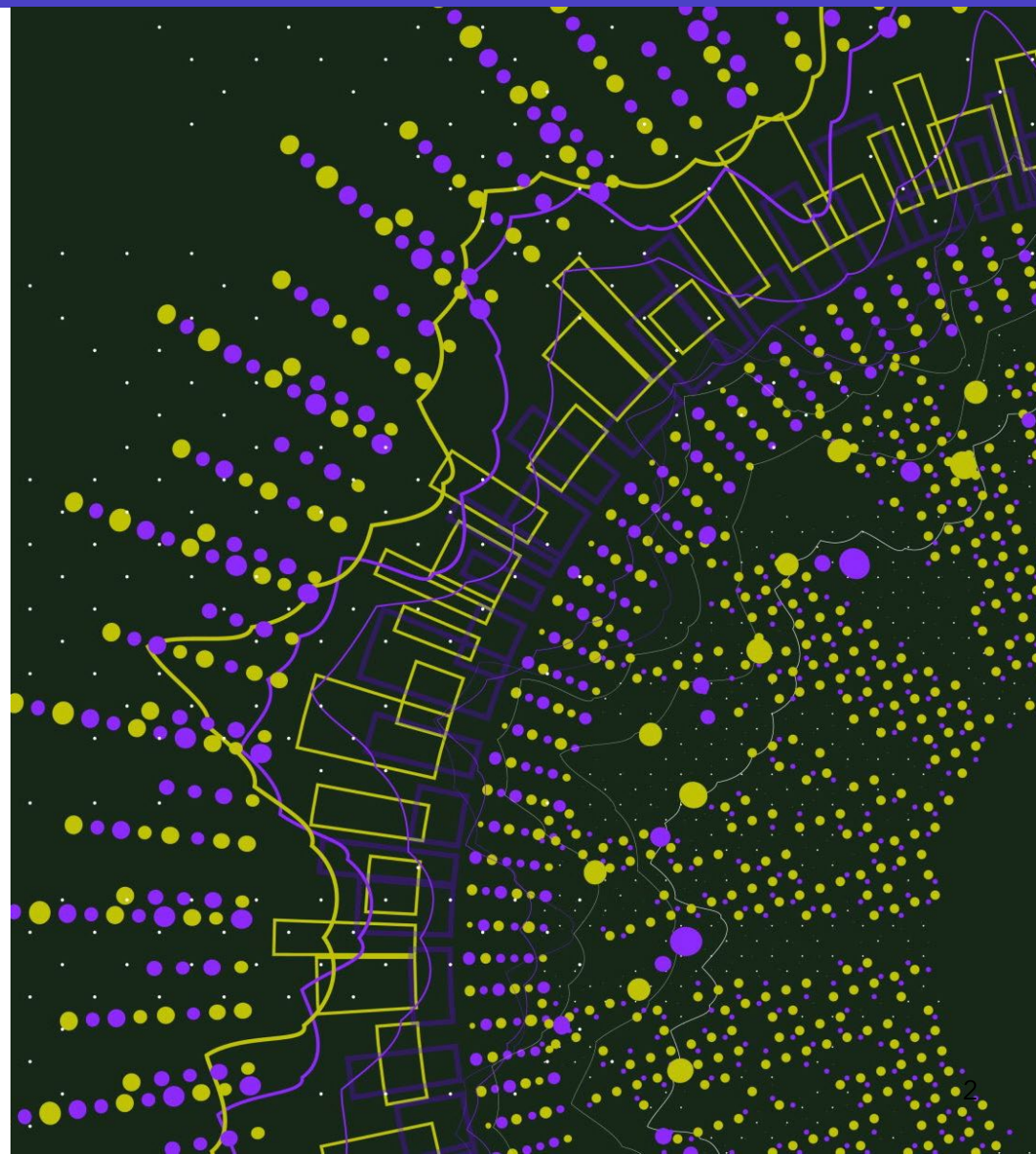
Supervised ML Linear Regression

Agenda

EDA Recap

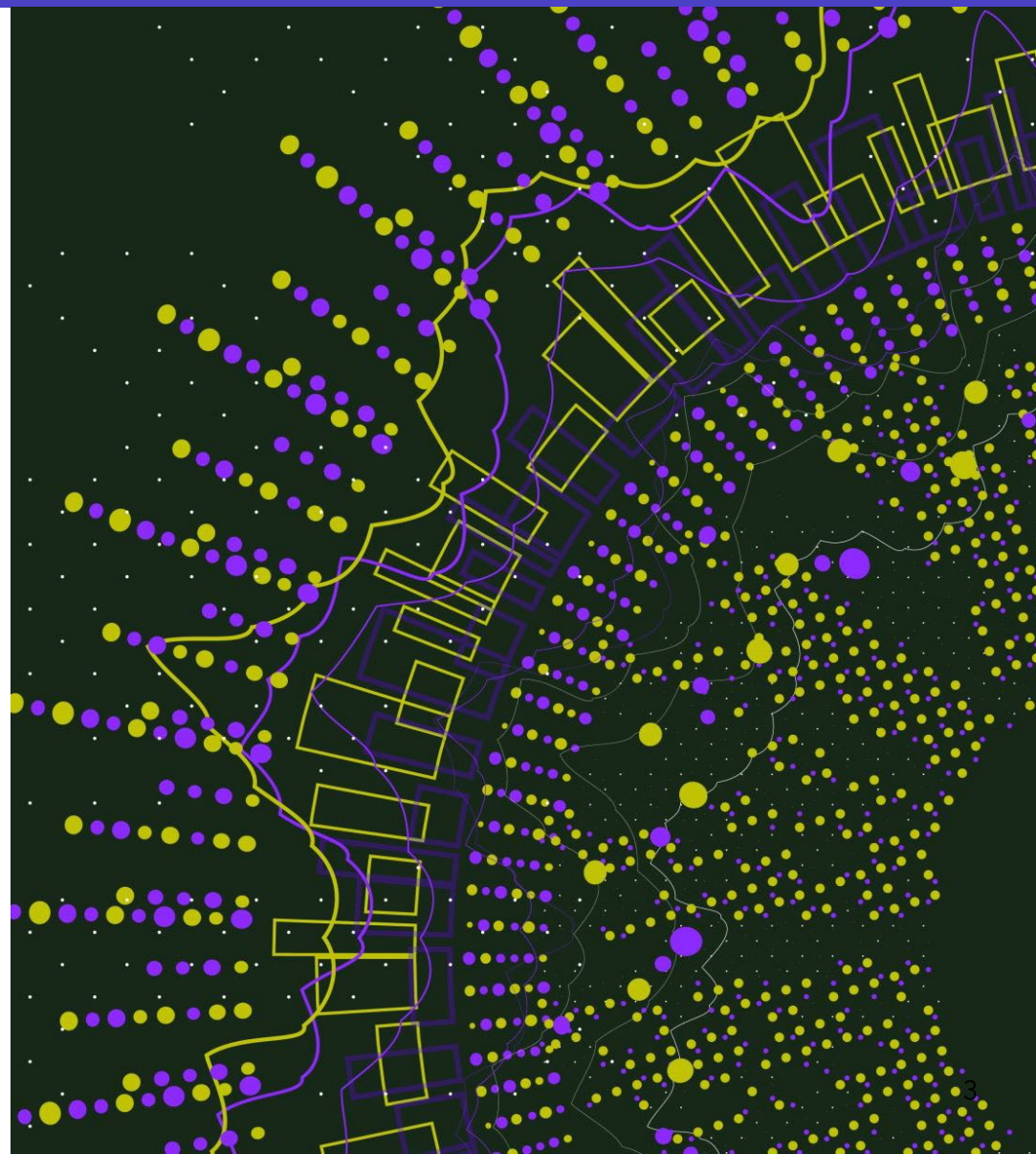
Model Training

Model Evaluation



EDA Recap

- Variable analyses
- Encoding categorical variables
- Imputation of missing values



Univariate Analysis

- Numerical Variables
  - Visualizations: Histogram, Box Plot
  - Summary Statistics: Mean, Median, Mode, Standard Deviation, Variance, Range, Interquartile Range (IQR)
  - `df.describe()`

- Categorical Variables:
  - Visualizations: Bar Plot, Pie Chart
  - Summary Statistics: Frequency Count, Proportion/Percentage
  - `df.Categories.value_counts(normalize=False, ascending=False)`

Bivariate Analysis

- Numerical vs. Numerical:
  - Visualizations: Scatter Plot, Correlation Matrix

```
import seaborn as sns
import matplotlib.pyplot as plt
# Scatter Plot
sns.scatterplot(x='variable1', y='variable2', data=df)
plt.show()

# Correlation Matrix
correlation_matrix = data.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm’)
plt.show()
```

Bivariate Analysis

- Numerical vs. Categorical:
  - Visualizations: Box Plot Count Plot
  - Summary Statistics: Group Mean/Median, Group Standard Deviation, Group Proportions

```
# Box Plot
sns.boxplot(x='categorical_var', y='numerical_var', data=df)
# Count Plot
sns.countplot(x='categorical_var', data=df)
# Group Summary Statistics
grouped_data = df.groupby('cat_var')['num_var'].agg(['mean', 'std'])
print(grouped_data)
```

Bivariate Analysis

- Categorical vs. Categorical: Contingency Table, Heatmap

```
# Contingency Table
contingency_table = pd.crosstab(df['cat_var1'],
df['cat_var2'])
print(contingency_table)

# Heatmap
sns.heatmap(contingency_table, annot=True, cmap='YlGnBu',
fmt='d')
plt.show()
```

Multivariate Analysis

- Visualizations:
  - Pair Plot
  - Multivariate Plot

```
# Pair Plot
sns.pairplot(df)

# Multivariate Plot (using Pair Grid)
g = sns.PairGrid(df)
g.map_upper(sns.scatterplot)
g.map_lower(sns.kdeplot)
g.map_diag(sns.histplot)
```

Encoding categorical variables

- Why?
- Label Encoding: Converts categories to numerical labels
  - Use Case: Ordinal variables where the order matters.
  - Example: ["Low", "Medium", "High"] → [0, 1, 2]
- One-Hot Encoding: Converts categories into binary columns.
  - Use Case: Nominal variables with no intrinsic order.
  - Example: ["Red", "Green", "Blue"] → [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
- Ordinal Encoding: Similar to label encoding but with a defined order.
  - Use Case: Ordinal variables where the order is crucial.
  - Example: ["Freshman", "Sophomore", "Junior", "Senior"] → [0, 1, 2, 3]

Encoding categorical variables

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OrdinalEncoder

# Label Encoding
label_encoder = LabelEncoder()
df['Category_Encoded'] = label_encoder.fit_transform(df['Category’])

# One-Hot Encoding
one_hot_encoded = pd.get_dummies(df, columns=['Color’])

# Ordinal Encoding
ordinal_encoder = OrdinalEncoder(categories=[['Small', 'Medium', 'Large']])
df['Size_Encoded'] = ordinal_encoder.fit_transform(df[['Size']])
```

Imputation of missing values

- Mean/Median/Mode Imputation: Replace missing values with the mean, median, or mode of the column
- Forward/Backward Fill: Use previous (forward fill) or next (backward fill) value to fill missing data for time series data
- K-Nearest Neighbors (KNN) Imputation: Use the values of K nearest neighbors to impute missing data
- Regression Imputation: Predict missing values using regression models based on other variables
- Indicator Variable: Add a binary indicator for missing values if a missing value is informative

Imputation of missing values

```
from sklearn.impute import SimpleImputer
from sklearn.impute import KNNImputer

# Mean Imputation
mean_imputer = SimpleImputer(strategy='mean’)
df['feature1’] = mean_imputer.fit_transform(df[['feature1']])

# Forward Fill
df['feature2'] = data['feature2'].fillna(method='ffill’)

# KNN Imputation
knn_imputer = KNNImputer(n_neighbors=5)
data_knn_imputed = knn_imputer.fit_transform(df)
```

Feature Selection

- Improve model performance by selecting relevant features
- Techniques
  - Domain knowledge
  - Statistical measures (e.g., correlation)
  - Model performance (e.g., recursive feature elimination)

```
X = df.drop('target', axis=1)
y = df['target']
```

Train-Test Split

- Split the dataset into training and testing subsets to train and evaluate model performance
- Commonly used ratios are 70-30, 80-20, or 90-10

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Linear Regression Implementation

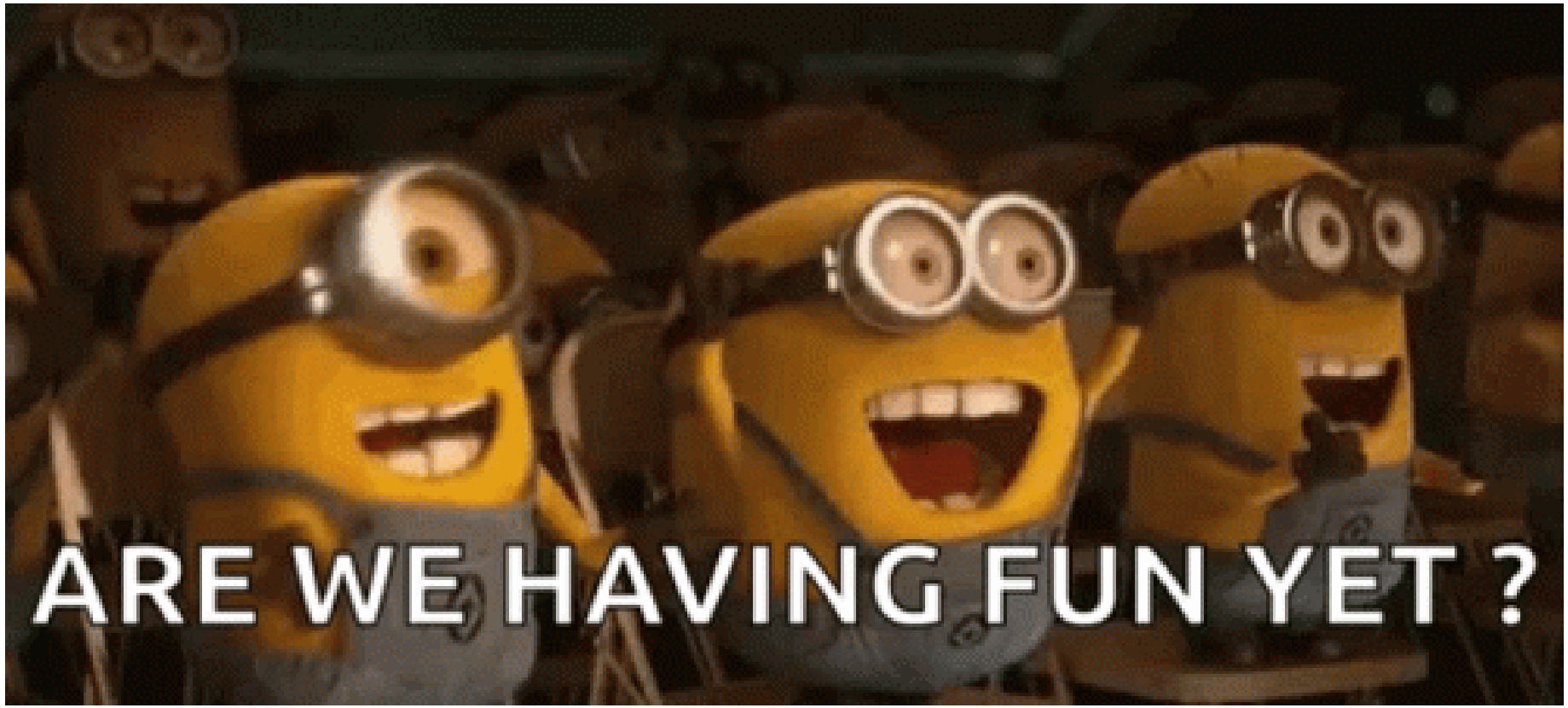
- Predict continuous target variables based on input features.
- Establishes a linear relationship between input variables (X) and output variable (y)

```
from sklearn.linear_model import LinearRegression

# Initialize and train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
```

Model Evaluation

- Assess the performance of the linear regression model on unseen data.
- Key Metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), R-squared (R^2),

```
from sklearn.metrics import mean_squared_error, r2_score,
mean_absolute_error
import numpy as np

# Evaluate model performance
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```



Let’s have some fun!