Module 3

Understanding Exploratory Data Analysis (EDA)

- EDA is a method for analyzing data sets to summarize their main characteristics, often using visual methods.
- It serves as an initial interaction with the data, helping to identify patterns, trends, and the need for data cleaning or additional data.

Techniques Used in EDA

- Common summary statistics include average, median, minimum, maximum, and correlations between columns.
- Visual techniques such as histograms, scatter plots, and box plots are used to understand data distribution and identify outliers.

Sampling from DataFrames

- Random sampling is useful for managing large data sets, allowing for efficient model training and testing.
- Stratified sampling ensures that the sample reflects the proportions of different observations in the data set, which is crucial for accurate analysis.

Code

```
# Sample 5 rows without replacement

sample = data.sample(n=5, replace=False)
print(sample.iloc[:,-3:])
```

Output

	petal_length	petal_width	species
73	4.7	1.2	Iris-versicolor
18	1.7	0.3	Iris-setosa
118	6.9	2.3	Iris-virginica
78	4.5	1.5	Iris-versicolor
76	4.8	1.4	Iris-versicolor

Matplotlib

- It is the primary library for creating plots and graphs in Python, offering flexibility and a wide range of features.
- The library requires a specific command (`%matplotlib inline`) to display plots in Jupyter notebooks.

Pandas and Seaborn

- Pandas provides a convenient wrapper around Matplotlib, allowing for easier plotting but with less flexibility.
- Seaborn, built on top of Matplotlib, simplifies the creation of aesthetically pleasing and statistically interesting plots.

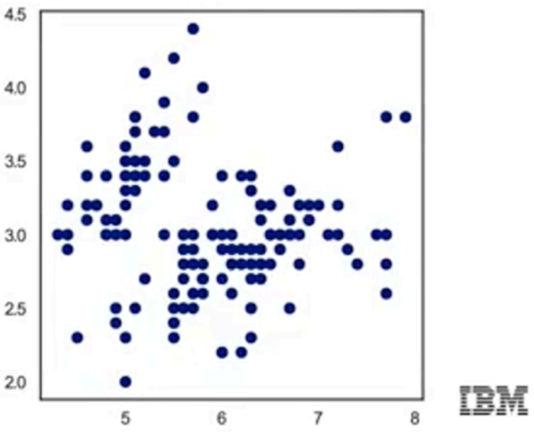
Creating Visualizations

- The lecture covers how to create scatter plots, histograms, and bar plots using both Matplotlib and Seaborn.
- It emphasizes the importance of customizing plots, such as setting labels, titles, and colors, to enhance clarity and presentation.

Code

```
# Pandas DataFrame approach
import matplotlib.pyplot as plt
plt.plot(data.sepal_length,
         data.sepal_width,
         ls ='', marker='o')
```

Output

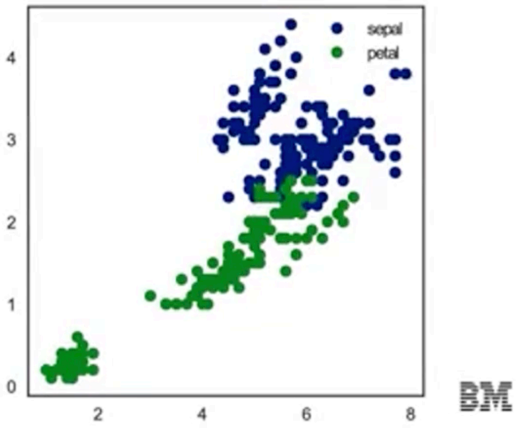


Code

```
# First plot statement
plt.plot(data.sepal_length,
         data.sepal_width,
         ls ='', marker='o',
         label='sepal')

# Second plot statement
plt.plot(data.petal_length,
         data.petal_width,
         ls ='', marker='o',
         label='petal')
```

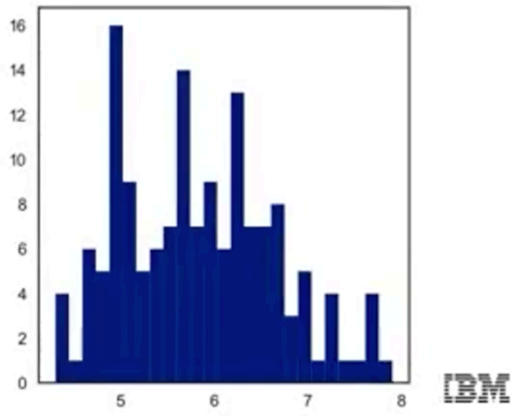
Output



Code

```
# Pandas DataFrame approach
plt.hist(data.sepal_length, bins=25)
```

Output

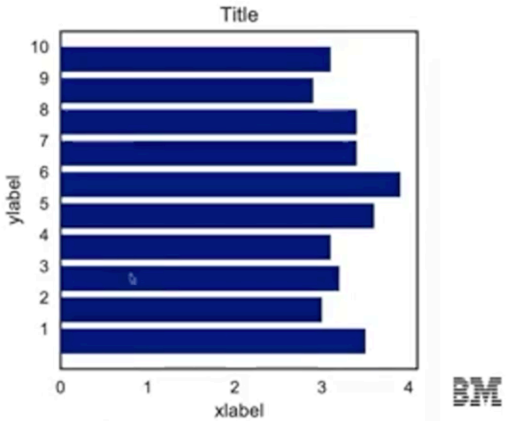


Code

```
# matplotlib syntax
fig, ax = plt.subplots()
ax.barh(np.arange(10),
        data.sepal_width.iloc[:10])

# Set position of ticks and tick labels
ax.set_yticks(np.arange(0.4,10.4,1.0))
ax.set_yticklabels(np.arange(1,11))
ax.set(xlabel='xlabel', ylabel='ylabel',
       title='Title')
```

Output



Panda Syntax for Plotting

- Group data by species and calculate the mean for features like petal length and width.
- Create a dot plot with specific colors for each feature, setting figure size and font size.

Seaborn Pair Plot

- Import Seaborn and use the pair plot function to visualize relationships between features.
- Set hue to species to differentiate data points by color, allowing for better analysis of correlations.

Hex Bin Plot and Facet Grid

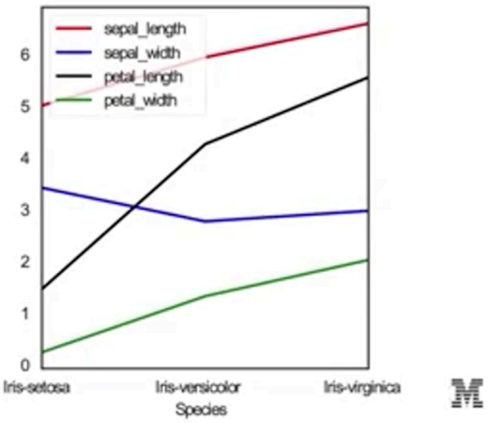
- Use sns.joint plot to create a hex bin plot, showing density of data points and histograms for additional insights.
- Implement a facet grid to visualize histograms for different species, enhancing the understanding of data distribution.

Overall, the section emphasizes various EDA techniques, including summary statistics and visualization methods, to better understand datasets before further analysis.

Code

```
# Pandas DataFrame approach
data.groupby('species').mean()
.plot(color=['red','blue',
      'black','green'],
      fontsize=10.0, figsize=(4,4))
```

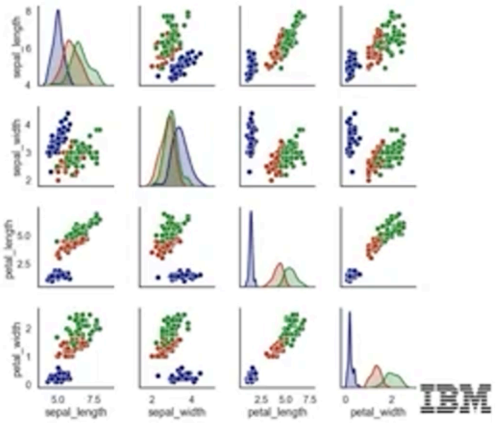
Output



Code

```
# Seaborn plot, feature correlations
sns.pairplot(data,
             hue='species', size=3)
```

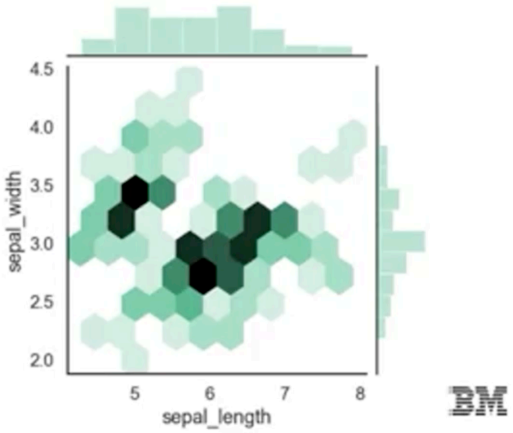
Output



Code

```
# Seaborn hexbin plot
sns.jointplot(x=data['sepal_length'],
              y=data['sepal_width'],
              kind='hex')
```

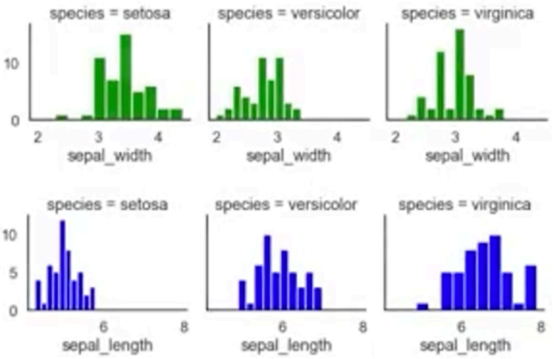
Output



Code

```
# Seaborn plot, Facet Grid
# First plot statement
plot = sns.FacetGrid(data,
                     col='species',
                     margin_titles=True)
plot.map(plt.hist, 'sepal_width',
         color='green')
# Second plot statement
plot = sns.FacetGrid(data,
                     col='species',
                     margin_titles=True)
plot.map(plt.hist, 'sepal_length',
         color='blue')
```

Output



Feature Engineering and Variable Transformation

- Feature engineering involves adjusting raw data to optimize model performance, including techniques like feature encoding and scaling.
- Variable transformation, such as log transformation, can help manage outliers and change data distribution.

Importance of Linear Relationships

- Many machine learning models, like linear regression, assume a linear relationship between predictor variables and the target outcome.
- Understanding how to represent these relationships mathematically is crucial for effective modeling, using parameters and coefficients to predict outcomes.

Transforming Data: Background

An example of a linear model relating (**feature**) variables x_1 and x_2 with target (**label**) variable y, is:

$$y_\beta(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

Here, $\beta = (\beta_0, \beta_1, \beta_2)$ represent the model's **parameters**.

Data Transformations

- Raw data can be skewed, affecting the performance of linear regression models.
- Transformations like log and Box-Cox can help normalize skewed data.

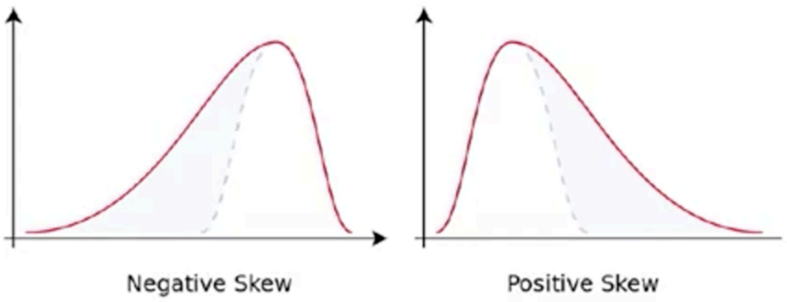
Log Transformation

- Log transformation can convert positively skewed data into a more normal distribution.
- It allows for linear relationships to be identified even when raw data does not exhibit linearity.

Polynomial Features

- Polynomial features can be added to the model to capture non-linear relationships.
- By including terms like x^2 or x^3, the model can represent more complex relationships while still being a linear regression model.

Data transformations can solve this issue.

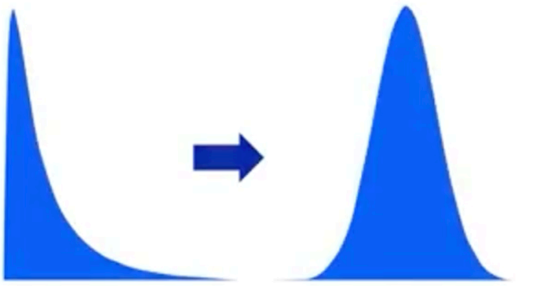


IBM

Code

```
# Useful transformation functions
from numpy import log, log1p
from scipy.stats import boxcox
```

Output

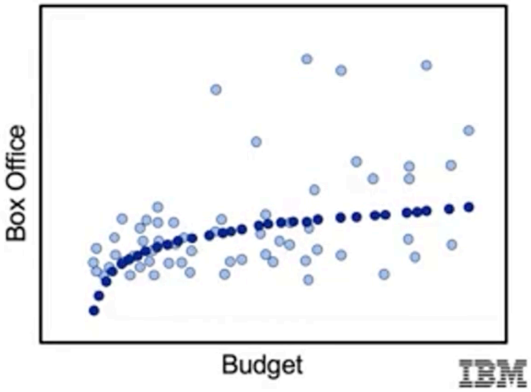


IBM

Log transformations can be useful for linear regression.

The linear regression model involves linear combinations of features.

$$y_\beta(x) = \beta_0 + \beta_1 \log(x)$$

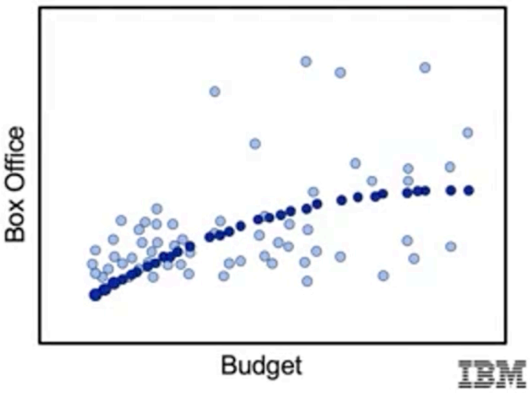


IBM

We can estimate higher-order relationships in this data by adding **polynomial features**.

This allows us to use the same 'linear' model.

$$y_\beta(x) = \beta_0 + \beta_1 x + \beta_2 x^2$$



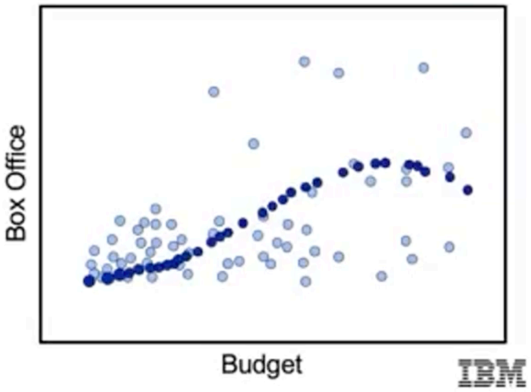
IBM

We can estimate higher-order relationships in this data by adding **polynomial features**.

This allows us to use the same 'linear' model.

Even with higher-order polynomials.

$$y_\beta(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$$



IBM

Polynomial Features: Syntax

Code

```
# Import the class containing the transformation method
from sklearn.preprocessing import PolynomialFeatures

# Create an instance of the class (choose number of degrees)
polyFeat = PolynomialFeatures(degree=2)

# Create the polynomial features and then transform the data
polyFeat = polyFeat.fit(X_data)
X_poly = polyFeat.transform(X_data)
```