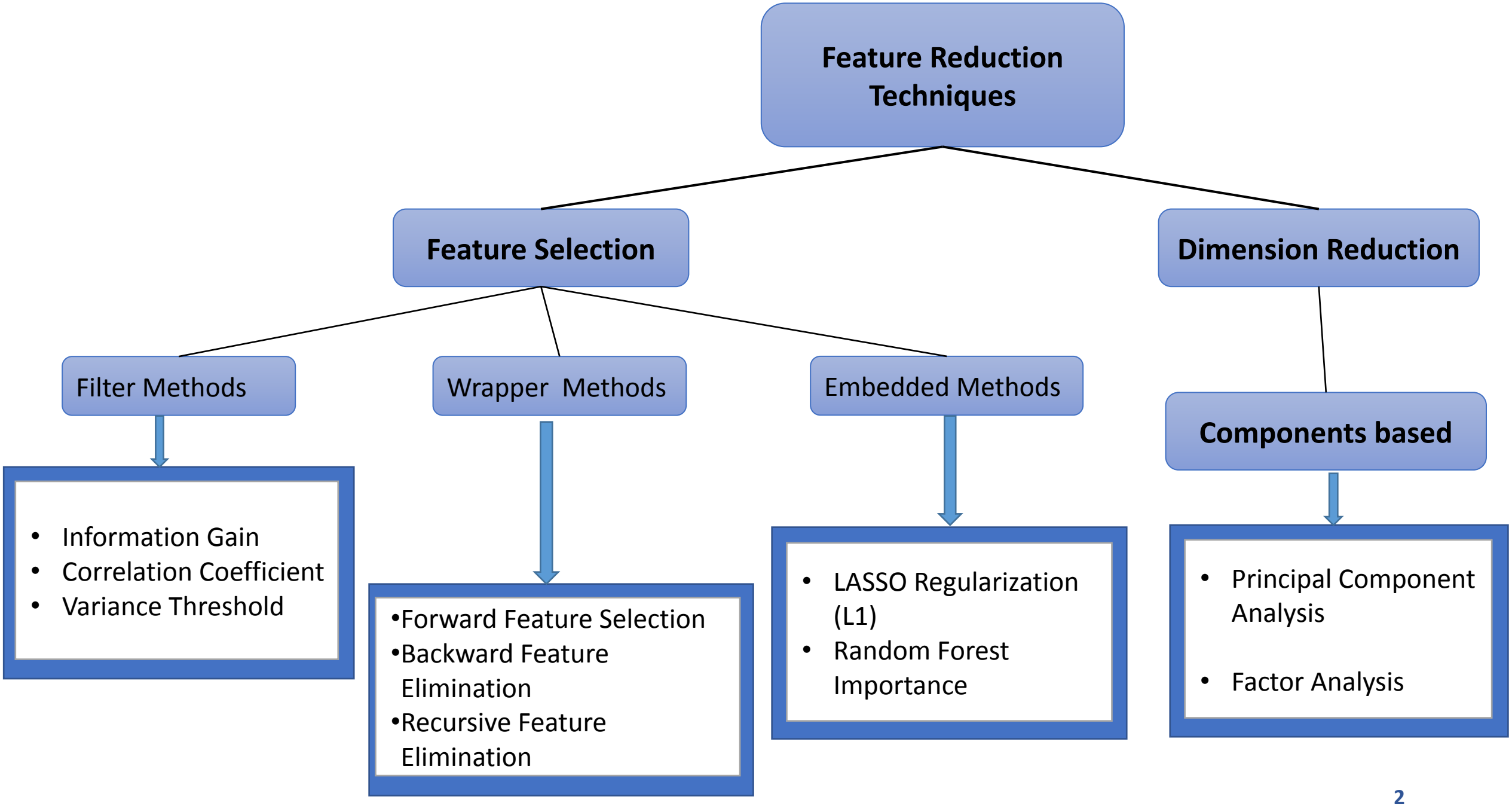
Data Visualization

Lecture 9

Data Preprocessing

Feature Reduction


Feature Reduction Techniques
Feature Selection
Dimension Reduction
Filter Methods
Wrapper Methods
Embedded Methods
Components based
• Information Gain
• Correlation Coefficient
• Variance Threshold
•Forward Feature Selection
•Backward Feature Elimination
•Recursive Feature Elimination
• LASSO Regularization (L1)
• Random Forest Importance
• Principal Component Analysis
• Factor Analysis

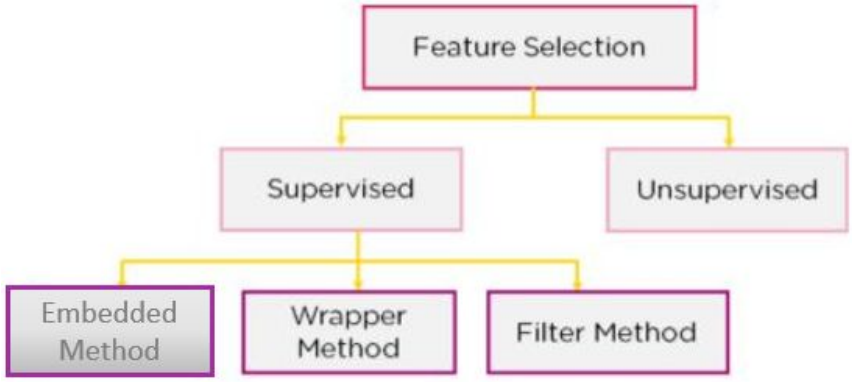
Feature Selection

What is Feature Selection?

- The goal of feature selection techniques in machine learning is to find the best set of features that allows one to build optimized models of studied phenomena.

What is Feature Selection?

- The techniques for feature selection in machine learning can be broadly classified into the following categories:
  - Supervised Techniques: These techniques can be used for labeled data and to identify the relevant features for increasing the efficiency of supervised models like classification and regression. For Example- linear regression, decision tree, SVM, etc.
  - Unsupervised Techniques: These techniques can be used for unlabeled data. For Example- K-Means Clustering, Principal Component Analysis, Hierarchical Clustering, etc.



Types of Feature Selection Methods in ML

Filter Methods

- Filter methods pick up the intrinsic properties of the features measured via univariate statistics instead of cross-validation performance. These methods are faster and less computationally expensive than wrapper methods. When dealing with high-dimensional data, it is computationally cheaper to use filter methods.

Wrapper Methods

- Wrappers require some method to search the space of all possible subsets of features, assessing their quality by learning and evaluating a classifier with that feature subset. The feature selection process is based on a specific machine learning algorithm we are trying to fit on a given dataset. It follows a greedy search approach by evaluating all the possible combinations of features against the evaluation criterion. The wrapper methods usually result in better predictive accuracy than filter methods.

Embedded Methods

- These methods encompass the benefits of both the wrapper and filter methods by including interactions of features but also maintaining reasonable computational costs. Embedded methods are iterative in the sense that takes care of each iteration of the model training process and carefully extract those features which contribute the most to the training for a particular iteration.

Feature Selection (Filter Methods)

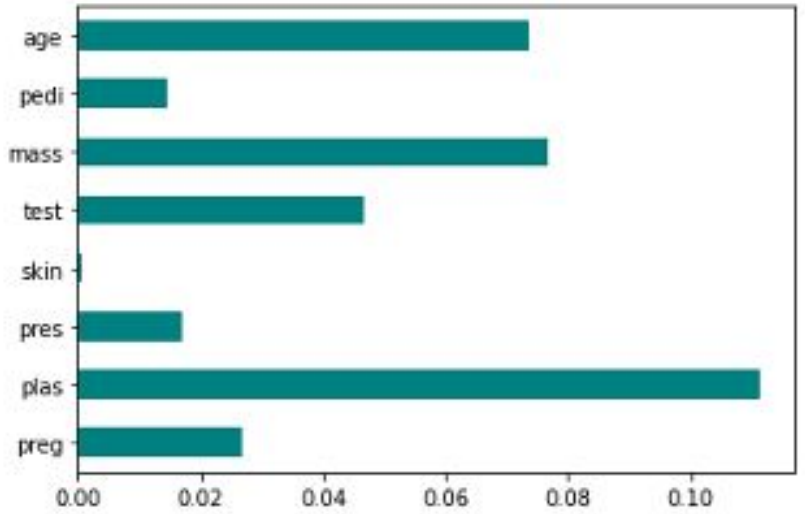
Feature Selection (Filter Methods)

- Feature Selection(Filter Methods):
  - Information Gain
  - Correlation Coefficient
  - Variance Threshold

Feature Selection (Information Gain Method)

- Information gain calculates the reduction in entropy (uncertainty) from the transformation of a dataset. It can be used for feature selection by evaluating the Information gain of each variable in the context of the target variable.

```
from sklearn.feature_selection import mutual_info_classif
import matplotlib.pyplot as plt
%matplotlib inline

importances = mutual_info_classif(X, Y)
feat_importances = pd.Series(importances, dataframe.columns[0:len(dataframe.columns)-1])
feat_importances.plot(kind='barh', color = 'teal')
plt.show()
```



Feature Selection (Correlation Coefficient Method)

- Correlation is a measure of the linear relationship between 2 or more variables. Through correlation, we can predict one variable from the other. The logic behind using correlation for feature selection is that good variables correlate highly with the target. Furthermore, variables should be correlated with the target but **uncorrelated among themselves.**

- If two variables are correlated, we can predict one from the other. Therefore, if two features are correlated, the model only needs one, as the second does not add additional information. We will use the Pearson Correlation here.

Feature Selection (Correlation Coefficient Method)

```
1  import seaborn as sns
2  import matplotlib.pyplot as plt
3  %matplotlib inline
4
5  # Correlation matrix
6  cor = dataframe.corr()
7
8  # Plotting Heatmap
9  plt.figure(figsize = (10,6))
10 sns.heatmap(cor, annot = True)
```

```
<AxesSubplot:>
```

Feature Selection(Variance Threshold Method)

- The variance threshold is a simple baseline approach to feature selection. It removes all features whose variance doesn't meet some threshold. By default, it removes all zero-variance features, i.e., features with the same value in all samples. We assume that features with a higher variance may contain more useful information but note that **we are not taking the relationship between feature variables or feature and target variables into account**, which is one of the drawbacks of filter methods.

```
1  from sklearn.feature_selection import VarianceThreshold
2
3  # Resetting the value of X to make it non-categorical
4  X = array[:,0:8]
5
6  v_threshold = VarianceThreshold(threshold=0)
7  v_threshold.fit(X)  # fit finds the features with zero variance
8  v_threshold.get_support()
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True])
```

Feature Selection (Wrapper Methods)

Feature Selection(Wrapper Methods)

- Feature Selection(Wrapper Methods):
  - Forward Feature Selection
  - Backward Feature Elimination
  - Recursive Feature Elimination

Feature Selection(Forward Feature Selection Method)

- This is an iterative method wherein we start with the performing features against the target features. Next, we select another variable that gives the best performance in combination with the first selected variable. This process continues until the preset criterion is achieved.

```
# Forward Feature Selection
from mlxtend.feature_selection import SequentialFeatureSelector
ffs = SequentialFeatureSelector(lr, k_features='best', forward = True, n_jobs=-1)
ffs.fit(X, Y)
features = list(ffs.k_feature_names_)
features = list(map(int, features))
lr.fit(x_train[features], y_train)
y_pred = lr.predict(x_train[features])
```

Feature Selection(Backward Feature Elimination Method)

- This method works exactly opposite to the Forward Feature Selection method. Here, we start with all the features available and build a model. Next, we the variable from the model, which gives the best evaluation measure value. This process is continued until the preset criterion is achieved.

```
# Backward Feature Selection
from sklearn.linear_model import LogisticRegression
from mlxtend.feature_selection import SequentialFeatureSelector
lr = LogisticRegression(class_weight = 'balanced', solver = 'lbfgs', random_state=42, n_jobs=-1, max_iter=500)
lr.fit(X, Y)
bfs = SequentialFeatureSelector(lr, k_features='best', forward = False, n_jobs=-1)
bfs.fit(X, Y)
features = list(bfs.k_feature_names_)
features = list(map(int, features))
lr.fit(x_train[features], y_train)
y_pred = lr.predict(x_train[features])
```

Feature Selection(Recursive Feature Elimination Method)

- Given an external estimator that assigns weights to features (e.g., the coefficients of a linear model), the goal of recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features, and each feature's importance is obtained either through a coef_ attribute or a feature_importances_ attribute.
- Then, the least important features are pruned from the current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.'

```
1  # Recursive Feature Selection
2  from sklearn.feature_selection import RFE
3  rfe = RFE(lr, n_features_to_select=7)
4  rfe.fit(x_train, y_train)
5  y_pred = rfe.predict(x_train)
```

Feature Selection (Embedded Methods)

General Steps

- **Initialization**: The process begins by assigning small, random values to all weights. These are initial guesses of how important the features might be.
- **Forward Propagation (Prediction)**: The model uses these initial weights to make a prediction for a given input.
- **Error Calculation (Loss Function)**: The model compares its prediction to the actual, correct answer (the target variable) using a **loss function**. This difference is the error.
- **Weight Adjustment (Optimization)**: The model then uses an optimization algorithm (like **Gradient Descent** and its variations) to figure out how to adjust the weights to *minimize* that error.
- **Iteration**: Steps 2-4 are repeated many times across the entire dataset until the model's error is minimized and the weights stabilize (converge).

Feature Selection(Embedded Methods)

- Feature Selection(Embedded Methods):
  - LASSO Regularization (L1)
  - Random Forest Importance

Feature Selection(LASSO Regularization (L1) Method)

- Regularization consists of adding a penalty to the different parameters of the machine learning model to reduce the freedom of the model, i.e., to avoid over-fitting. In linear model regularization, the penalty is applied over the coefficients that multiply each predictor. From the different types of regularization, Lasso or L1 has the property that can shrink some of the coefficients to zero. Therefore, that feature can be removed from the model.

```
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectFromModel

# Set the regularization parameter C=1
logistic = LogisticRegression(C=1, penalty="l1", solver='liblinear', random_state=7).fit(X, Y)
model = SelectFromModel(logistic, prefit=True)

X_new = model.transform(X)

# Dropped columns have values of all 0s, keep other columns
selected_columns = selected_features.columns[selected_features.var() != 0]
selected_columns
```

```
Int64Index([0, 1, 2, 3, 4, 5, 6, 7], dtype='int64')
```

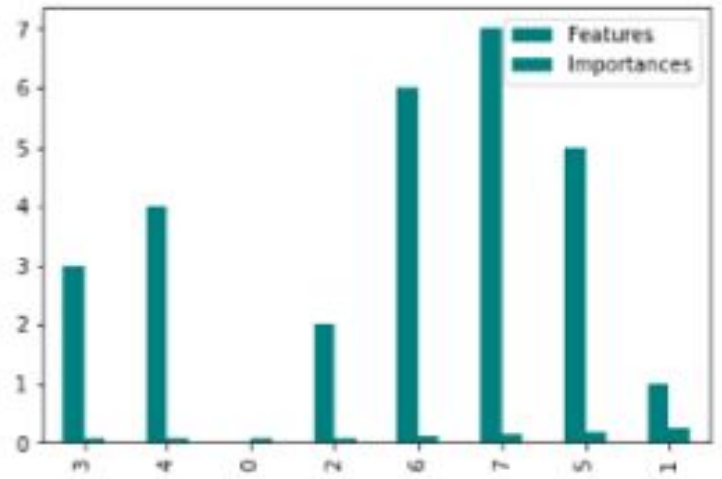
Feature Selection(Random Forest Importance Method)

- Random Forests is a kind of Bagging Algorithm that aggregates a specified number of decision trees. The tree-based strategies used by random forests naturally rank by how well they improve the purity of the node, or in other words, a decrease in the impurity (Gini impurity) over all trees. Nodes with the greatest decrease in impurity happen at the start of the trees, while notes with the least decrease in impurity occur at the end of the trees. Thus, by pruning trees below a particular node, we can create a subset of the most important features.

Feature Selection(Random Forest Importance Method)

```python
from sklearn.ensemble import RandomForestClassifier

# create the random forest with your hyperparameters.
model = RandomForestClassifier(n_estimators=340)

# fit the model to start training.
model.fit(X, Y)

# get the importance of the resulting features.
importances = model.feature_importances_

# create a data frame for visualization.
final_df = pd.DataFrame({"Features": pd.DataFrame(X).columns, "Importances":importances})
final_df.set_index('Importances')

# sort in ascending order to better visualization.
final_df = final_df.sort_values('Importances')

# plot the feature importances in bars.
final_df.plot.bar(color = 'teal')
```

```
<AxesSubplot:>
```



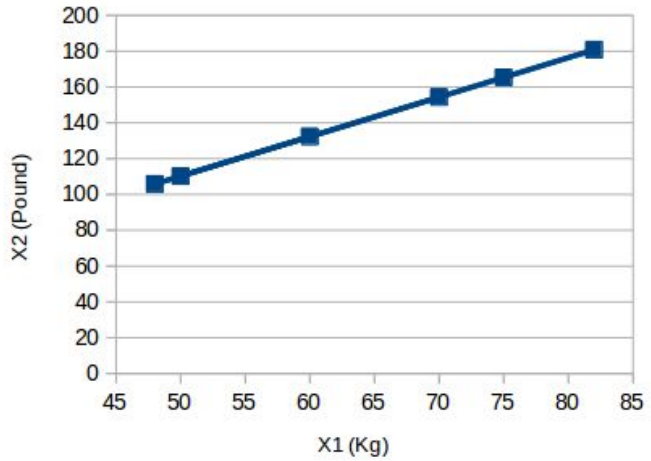
Dimensionality Reduction

What Is Dimensionality Reduction?

- As data generation and collection keeps increasing, visualizing it and drawing inferences becomes more and more challenging.
- Suppose we have 2 variables, Age and Height. We can use a scatter or line plot between Age and Height and visualize their relationship easily
- Now consider a case in which we have, say 100 variables (p=100). In this case, we can have 100(100-1)/2 = 5000 different plots. It does not make much sense to visualize each of them separately, right? In such cases where we have a large number of variables, it is better to select a subset of these variables (p<<100) which captures as much information as the original set of variables.

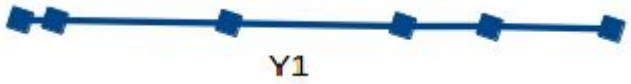
What Is Dimensionality Reduction?

- Let us understand this with a simple example. Consider the below image:



Similarly, we can reduce p dimensions of the data into a subset of k dimensions (k<<n). This is called dimensionality reduction.

- Here we have weights of similar objects in Kg (X1) and Pound (X2). If we use both of these variables, they will convey similar information. So, it would make sense to use only one variable. We can convert the data from 2D (X1 and X2) to 1D (Y1) as shown below:



Why is Dimensionality Reduction required?

- **Here are some of the benefits of applying dimensionality reduction to a dataset:**
  - Space required to **store** the data is reduced as the number of dimensions comes down
  - Less dimensions lead to less **computation/training time**
  - Some algorithms do not perform well when we have a large dimensions. So, reducing these dimensions needs to happen for the algorithm to be useful
  - It takes care of **multicollinearity** by removing redundant features. For example, you have two variables – 'time spent on treadmill in minutes' and 'calories burnt'. These variables are highly correlated as the more time you spend running on a treadmill, the more calories you will burn. Hence, there is no point in storing both as just one of them does what you require
  - It helps in **visualizing** data. As discussed earlier, it is very difficult to visualize data in higher dimensions so reducing our space to 2D or 3D may allow us to plot and observe patterns more clearly

Dimension Reduction Techniques

- Dimension Reduction Techniques:
  - Principal Component Analysis
  - Factor Analysis

Dimension Reduction (Principal Component Analysis)

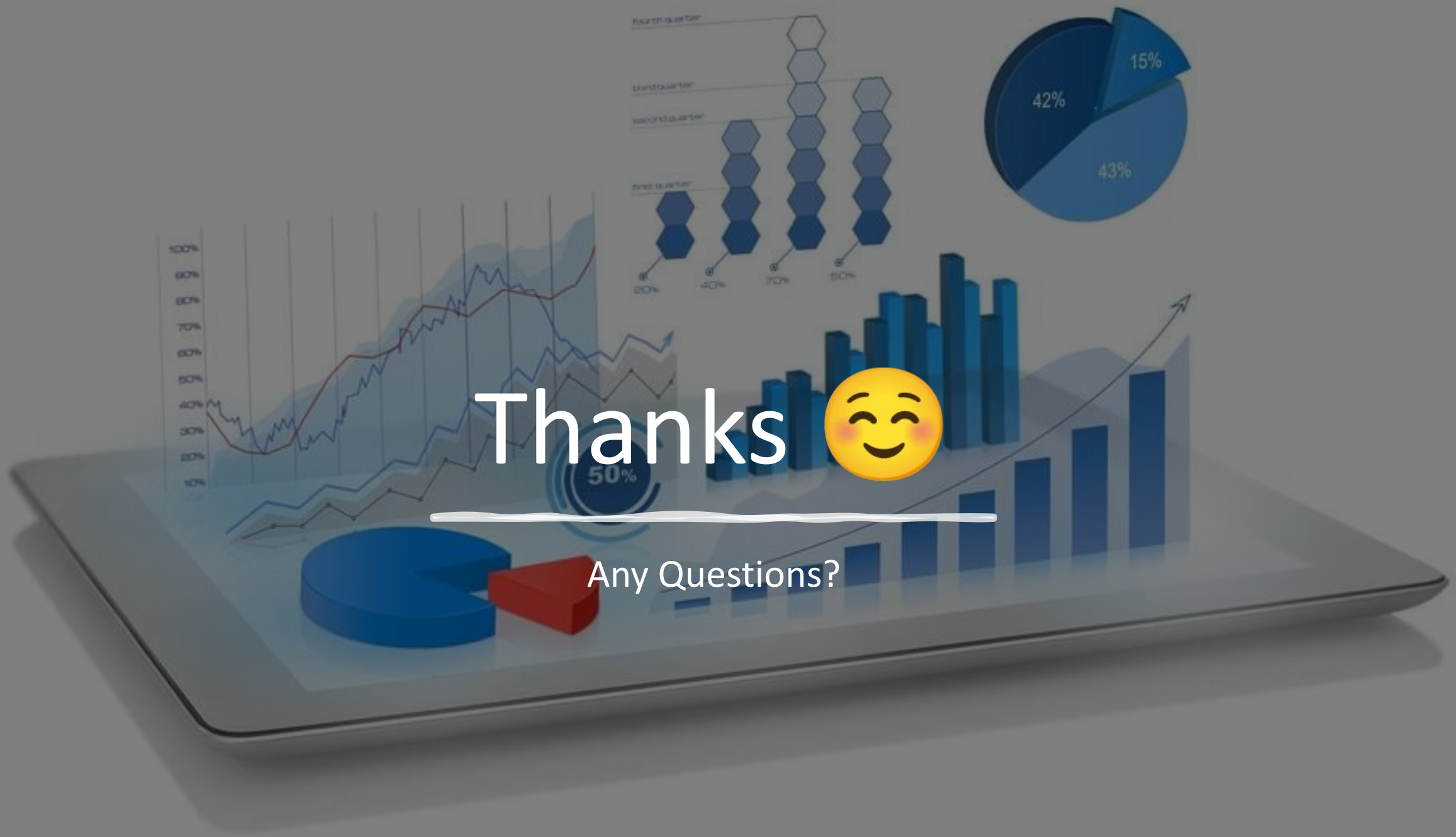
- PCA is a technique which helps us in extracting a new set of variables from an existing large set of variables. These newly (deriving) extracted variables are called Principal Components.
- Principal Component Analysis (PCA) is the technique that removes dependency or redundancy in the data by dropping those features that contain the same information as given by other attributes. and the derived components are independent of each other.
- These components are a linear combination of the original variables. This way, PCA converts a larger number of correlated variables (i.e., breaks down the data) into a smaller set of uncorrelated variables.

Dimension Reduction (Principal Component Analysis)

- below are some of the key points you should know about PCA before proceeding further:
  - A principal component is a linear combination of the original variables
  - Principal components are extracted in such a way that the first principal component explains maximum variance in the dataset
  - Second principal component tries to explain the remaining variance in the dataset and is uncorrelated to the first principal component
  - Third principal component tries to explain the variance which is not explained by the first two principal components and so on

Dimension Reduction (Principal Component Analysis)

- The steps involved in the process of performing PCA or FA are same:
  - Standardization of the predictors
  - Generation of the correlation matrix for these standardized values
  - Decomposition of the correlation matrix to eigenvectors and finding their respective eigenvalues
  - Rank the vectors in descending order of their eigenvalues



Thanks 😊

Any Questions?