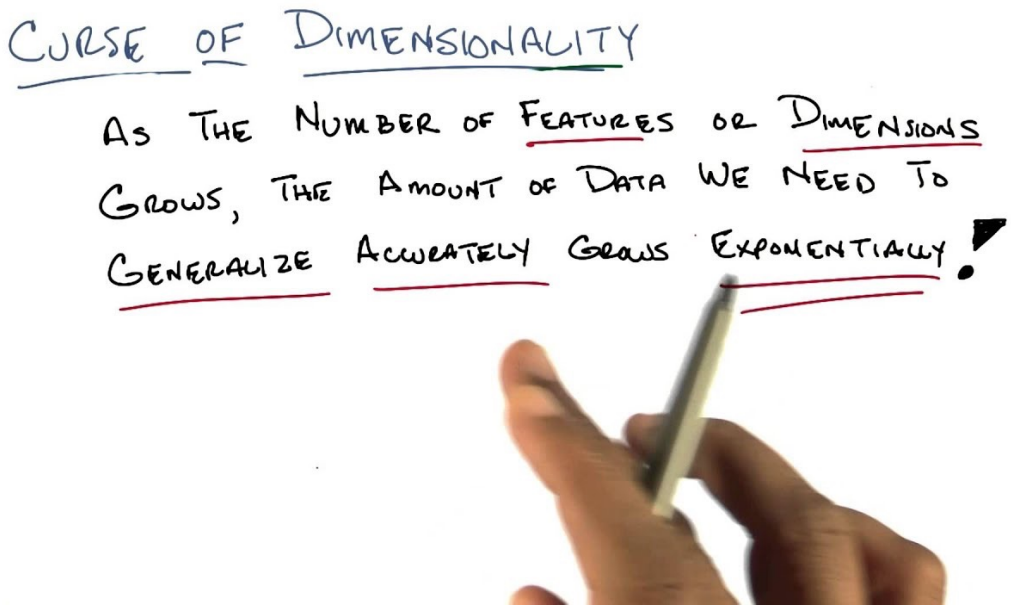
# Dimensionality Reduction Techniques

# Curse of Dimensionality

... refers to the difficulties that arise when analyzing or modeling data with many dimensions.

Data points become increasingly spread out (Data Sparsity), making it hard to find patterns or relationships.



# Three simple techniques to reduce dimensionality

- **Missing values ratio**

  If the percentage of missing values in a variable exceeds the threshold (a pre-specified value), you can drop the variable.

- **Low variance filter**

  All the data columns with variance lower than the threshold value will be eliminated.

- **High correlation filter**

  All the pairs of columns having a correlation coefficient higher than the set threshold will be reduced to 1.

# Principal Component Analysis (PCA)

It is a statistical method used to simplify and summarize data by reducing its dimensionality while retaining as much of the original variability as possible.

1. Apply PCA to the predictors using the training data. Use the output to determine the number of principal components to be retained. The predictors in the model now use the (reduced number of) principal scores columns. For the validation set, we can use the weights computed from the training data to obtain a set of principal scores by applying the weights to the variables in the validation set. These new variables are then treated as the predictors.

2. One **disadvantage** of using a subset of principal components as predictors in a supervised task, is that we might lose predictive information that is nonlinear (e.g., a quadratic effect of a predictor on the outcome variable or an interaction between predictors). This is because PCA produces linear transformations, thereby capturing linear relationships between the original variables.

# Cereals

| Variable | Description |
|---|---|
| mfr | Manufacturer of cereal (American Home Food Products, General Mills, Kellogg, etc.) |
| type | Cold or hot |
| calories | Calories per serving |
| protein | Grams of protein |
| fat | Grams of fat |
| sodium | Milligrams of sodium |
| fiber | Grams of dietary fiber |
| carbo | Grams of complex carbohydrates |
| sugars | Grams of sugars |
| potass | Milligrams of potassium |
| vitamins | Vitamins and minerals: 0, 25, or 100, indicating the typical percentage of FDA recommended |
| shelf | Display shelf (1, 2, or 3, counting from the floor) |
| weight | Weight in ounces of one serving |
| cups | Number of cups in one serving |
| rating | Rating of the cereal calculated by *consumer reports* |

# Cereals

| | name | mfr | type | calories | protein | fat | sodium | fiber | carbo | sugars | potass | vitamins | shelf | weight | cups | rating |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 100%_Bran | N | C | 70 | 4 | 1 | 130 | 10.0 | 5.0 | 6.0 | 280.0 | 25 | 3 | 1.0 | 0.33 | 68.402973 |
| 1 | 100%_Natural_Bran | Q | C | 120 | 3 | 5 | 15 | 2.0 | 8.0 | 8.0 | 135.0 | 0 | 3 | 1.0 | 1.00 | 33.983679 |
| 2 | All-Bran | K | C | 70 | 4 | 1 | 260 | 9.0 | 7.0 | 5.0 | 320.0 | 25 | 3 | 1.0 | 0.33 | 59.425505 |
| 3 | All-Bran_with_Extra_Fiber | K | C | 50 | 4 | 0 | 140 | 14.0 | 8.0 | 0.0 | 330.0 | 25 | 3 | 1.0 | 0.50 | 93.704912 |
| 4 | Almond_Delight | R | C | 110 | 2 | 2 | 200 | 1.0 | 14.0 | 8.0 | NaN | 25 | 3 | 1.0 | 0.75 | 34.384843 |

```
pcs = PCA()
pcs.fit(preprocessing.scale(cereals_df.iloc[:, 3:15].dropna(axis=0)))
pcsSummary_df = pd.DataFrame({'Standard deviation': np.sqrt(pcs.explained_variance_),
                    'Proportion of variance': pcs.explained_variance_ratio_,
                    'Cumulative proportion': np.cumsum(pcs.explained_variance_ratio_)})
pcsSummary_df = pcsSummary_df.transpose()
pcsSummary_df.columns = ['PC{}'.format(i) for i in range(1, len(pcsSummary_df.columns) + 1)]
pcsSummary_df.round(4)
```

SKLEARN
**PCA()**

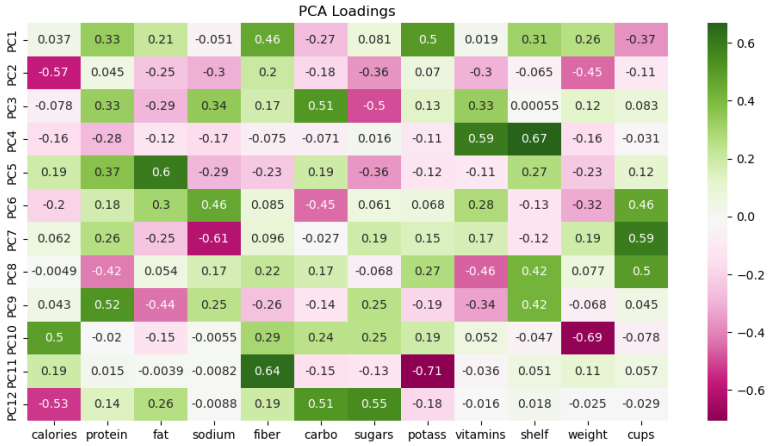
| | PC1 | PC2 | PC3 | PC4 | PC5 | PC6 | PC7 | PC8 | PC9 | PC10 | PC11 | PC12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Standard deviation** | 1.8344 | 1.6381 | 1.3455 | 1.0156 | 0.9989 | 0.8362 | 0.8179 | 0.6494 | 0.5639 | 0.3050 | 0.2389 | 0.1353 |
| **Proportion of variance** | 0.2766 | 0.2206 | 0.1488 | 0.0848 | 0.0820 | 0.0575 | 0.0550 | 0.0347 | 0.0261 | 0.0076 | 0.0047 | 0.0015 |
| **Cumulative proportion** | 0.2766 | 0.4972 | 0.6460 | 0.7308 | 0.8129 | 0.8704 | 0.9253 | 0.9600 | 0.9862 | 0.9938 | 0.9985 | 1.0000 |

# Cereals

```
pcsComponents_df = pd.DataFrame(pcs.components_.transpose(), columns=pcsSummary_df.columns,
                                index=cereals_df.iloc[:, 3:15].columns)
pcsComponents_df.iloc[:,:5]
```

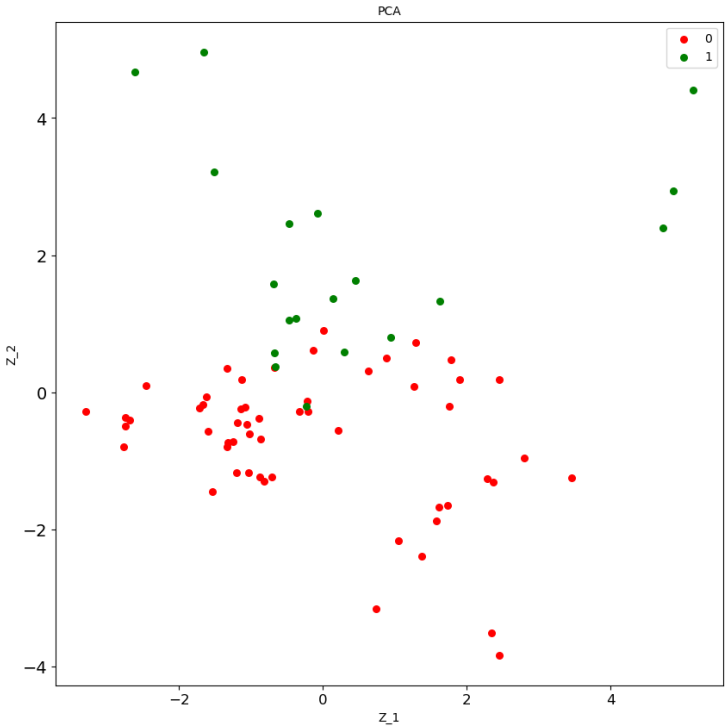
|: 

| | PC1 | PC2 | PC3 | PC4 | PC5 |
|---|---|---|---|---|---|
| **calories** | 0.036594 | -0.571142 | -0.077927 | -0.164032 | 0.186667 |
| **protein** | 0.330888 | 0.045367 | 0.326405 | -0.281265 | 0.372134 |
| **fat** | 0.212549 | -0.249143 | -0.291274 | -0.124324 | 0.597422 |
| **sodium** | -0.050894 | -0.304684 | 0.343108 | -0.168916 | -0.292344 |
| **fiber** | 0.458623 | 0.201029 | 0.173912 | -0.075295 | -0.229653 |
| **carbo** | -0.268609 | -0.175660 | 0.513002 | -0.071305 | 0.191830 |
| **sugars** | 0.081440 | -0.364844 | -0.495338 | 0.016033 | -0.360371 |
| **potass** | 0.502603 | 0.070345 | 0.130460 | -0.112708 | -0.124888 |
| **vitamins** | 0.018839 | -0.300574 | 0.327563 | 0.594798 | -0.105582 |
| **shelf** | 0.308929 | -0.065114 | 0.000546 | 0.667703 | 0.265485 |
| **weight** | 0.257160 | -0.448962 | 0.122030 | -0.160396 | -0.232905 |
| **cups** | -0.371154 | -0.106553 | 0.082695 | -0.030683 | 0.115826 |




It is difficult to give an interpretation to the Principal Components

# Cereals




When the data can be reduced to two dimensions, a useful plot is a scatter plot of the first vs. second principal scores with labels for the observations

APPLIED BUSINESS ANALYTICS

# Performance Evaluation

# Predictive Power Assessment

# Predictive Performance Evaluation

**Predicted numerical value**
*when the outcome variable is numerical*

**Predicted class membership**
*when the outcome variable is categorical*

**Propensity - the probability of class membership**
*when the outcome variable is categorical*

# Prediction Accuracy Measures

The **prediction error** is defined as the difference between its actual outcome value and its predicted outcome value: $e_i = y_i - \hat{y}_i$.

- **Mean Absolute Error** (MAE) $= \frac{1}{n}\sum|e_i|$

  the magnitude of the average absolute error.

- **Mean Absolute Percentage Error** (MAPE) $= 100 \times \frac{1}{n}\sum\left|\frac{e_i}{y_i}\right|$

  a percentage score of how predictions deviate (on average) from the actual values.

- **Root Mean Squared Error** (RMSE) $= \sqrt{\frac{1}{n}\sum e_i^2}$

  is similar to the standard error.

# Toyota Car Prices

```
car_df = pd.read_csv('ToyotaCorolla.csv')

# create a list of predictor variables by removing output variables and text columns
excludeColumns = ('Price', 'Id', 'Model', 'Fuel_Type', 'Color')
predictors = [s for s in car_df.columns if s not in excludeColumns] # the explanatory variables
outcome = 'Price' # the dependent variable

# partition data
X = car_df[predictors]
y = car_df[outcome]
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.4,random_state=1)
# train linear regression model
reg = LinearRegression()
reg.fit(train_X, train_y)
```

| | Id | Model | Price | Age_08_04 | Mfg_Month | Mfg_Year | KM | Fuel_Type | HP | Met_Color | ... | Powered_Windows | Power_Steering | Radio | Mistlamps | Sp |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors | 13500 | 23 | 10 | 2002 | 46986 | Diesel | 90 | 1 | ... | 1 | 1 | 0 | 0 | |
| 1 | 2 | TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors | 13750 | 23 | 10 | 2002 | 72937 | Diesel | 90 | 1 | ... | 0 | 1 | 0 | 0 | |
| 2 | 3 | TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors | 13950 | 24 | 9 | 2002 | 41711 | Diesel | 90 | 1 | ... | 0 | 1 | 0 | 0 | |
| 3 | 4 | TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors | 14950 | 26 | 7 | 2002 | 48000 | Diesel | 90 | 0 | ... | 0 | 1 | 0 | 0 | |
| 4 | 5 | TOYOTA Corolla 2.0 D4D HATCHB SOL 2/3-Doors | 13750 | 30 | 3 | 2002 | 38500 | Diesel | 90 | 0 | ... | 1 | 1 | 0 | 1 | |

**SKLEARN**

**LinearRegression()**

# Toyota Car Prices

```python
# evaluate performance
# training
mae = metrics.mean_absolute_error(train_y, reg.predict(train_X))
mape =metrics.mean_absolute_percentage_error(train_y, reg.predict(train_X))
mse = metrics.mean_squared_error(train_y, reg.predict(train_X))
rmse = np.sqrt(mse)
print('Performance Measures - Training set')
print(f'Mean absolute error: {mae:.2f}')
print(f'Mean absolute percentage error: {mape:.2f}')
print(f'Root mean squared error: {rmse:.2f}')

# validation
mae = metrics.mean_absolute_error(valid_y, reg.predict(valid_X))
mape =metrics.mean_absolute_percentage_error(valid_y, reg.predict(valid_X))
mse = metrics.mean_squared_error(valid_y, reg.predict(valid_X))
rmse = np.sqrt(mse)
print('\nPerformance Measures - Validation set')
print(f'Mean absolute error: {mae:.2f}')
print(f'Mean absolute percentage error: {mape:.2f}')
print(f'Root mean squared error: {rmse:.2f}')
```

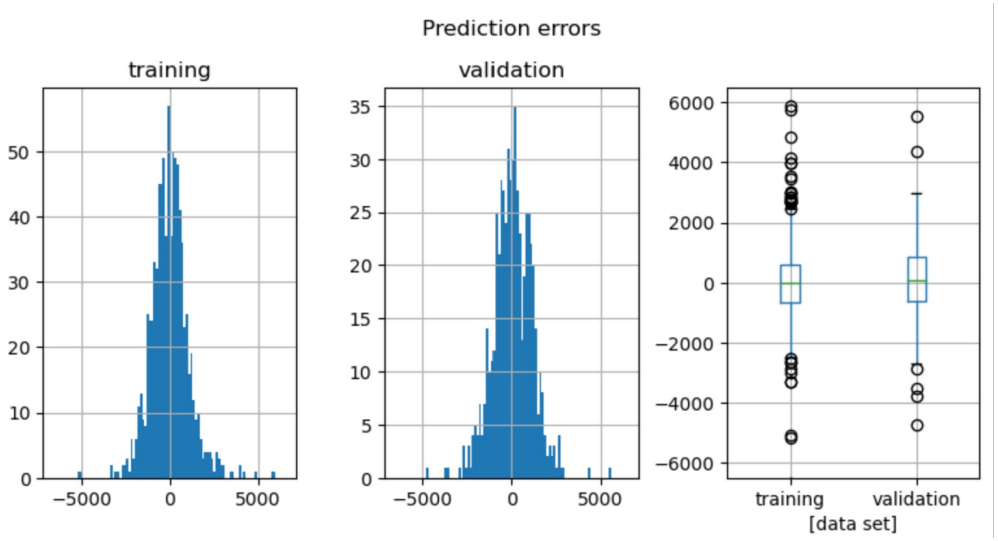
**SKLEARN**

**LinearRegression()**

```
Performance Measures - Training set
Mean absolute error: 811.68
Mean absolute percentage error: 0.08
Root mean squared error: 1121.06

Performance Measures - Validation set
Mean absolute error: 880.14
Mean absolute percentage error: 0.09
Root mean squared error: 1382.04
```

# Toyota Car Prices




```
pred_error_train = pd.DataFrame({
    'residual': train_y - reg.predict(train_X),
    'data set': 'training'
})
pred_error_valid = pd.DataFrame({
    'residual': valid_y - reg.predict(valid_X),
    'data set': 'validation'
})
boxdata_df = pd.concat([pred_error_train, pred_error_valid], ignore_index=True)

fig, axes = plt.subplots(nrows=1, ncols=3)
fig.set_size_inches(9, 4)
common = {'bins': 100, 'range': [-6500, 6500]}
pred_error_train.hist(ax=axes[0], **common)
pred_error_valid.hist(ax=axes[1], **common)
boxdata_df.boxplot(ax=axes[2], by='data set')

axes[0].set_title('training')
axes[1].set_title('validation')
axes[2].set_title(' ')
axes[2].set_ylim(-6500, 6500)
plt.suptitle('Prediction errors')
plt.subplots_adjust(bottom=0.1, top=0.85, wspace=0.35)
plt.show()
```

# Judging Classifier Performance

# Confusion Matrix

A **confusion matrix** is a matrix that summarizes the performance of a machine learning model on a set of test data. It is a means of displaying the number of accurate and inaccurate instances based on the model's predictions. It is often used to measure the performance of classification models, which aim to predict a categorical label for each input instance.

| | | Actual Value | |
|---|---|---|---|
| | | POSITIVE | NEGATIVE |
| Predicted Value | POSITIVE | TP | FP |
| | NEGATIVE | FN | TN |

# Accuracy Measures

- **Accuracy** $= \frac{TP+TN}{TP+FT+FN+TN}$

  fraction of predictions our model got right.

- **Precision** $= \frac{TP}{TP+FP}$

  how many of the positive class samples present in the dataset were correctly identified by the model.

- **Recall** $= \frac{TP}{TP+FN}$

  how many of the “positive” predictions made by the model were correct.

# Accuracy Measures

- $\mathbf{F1-score} = \frac{2 \times Precision \times Recall}{Precision + Recall}$

The F1 score combines precision and recall using their harmonic mean, and maximizing the F1 score implies simultaneously maximizing both precision and recall.

# Performance in Case of Unequal Importance of Classes

The **sensitivity** of a classifier is its ability to detect the important class members correctly. This is the recall and is the percentage of C1 members classified correctly.

The **specificity** of a classifier is its ability to rule out C2 members correctly. This is measured by the recall of C2 and is the percentage of C2 members classified correctly.

| | | Actual Value | |
|---|---|---|---|
| | | CLASS 1 | CLASS 2 |
| Predicted Value | CLASS 1 | $n_{1,1}$ | $n_{2,1}$ |
| | CLASS 2 | $n_{1,2}$ | $n_{2,2}$ |

# ROC Curve

A **ROC (Receiver Operating Characteristic) curve** is a plot of the true positive rate (Sensitivity) in function of the false positive rate (100 - Specificity) for different cut-off points of a parameter. Each point on the ROC curve represents a sensitivity/specificity pair corresponding to a particular decision threshold. The **Area Under the ROC curve (AUC)** is a measure of how well a parameter can distinguish between two classes.