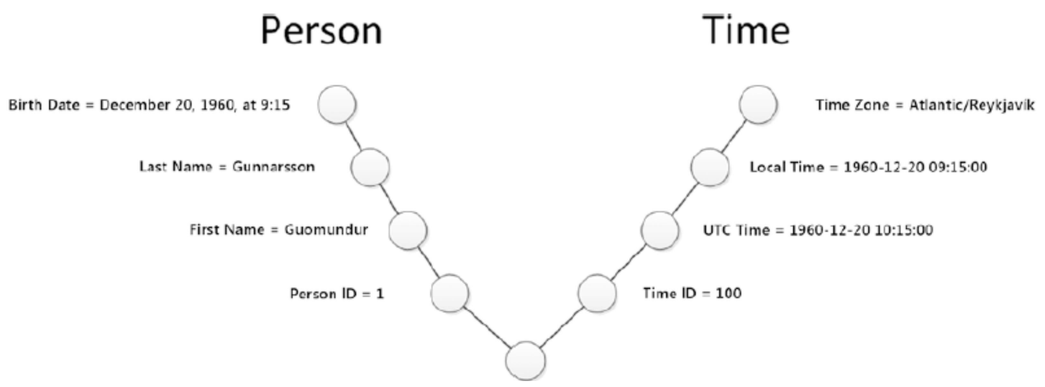
Unit 4 : Chapter 2: Transform Superstep

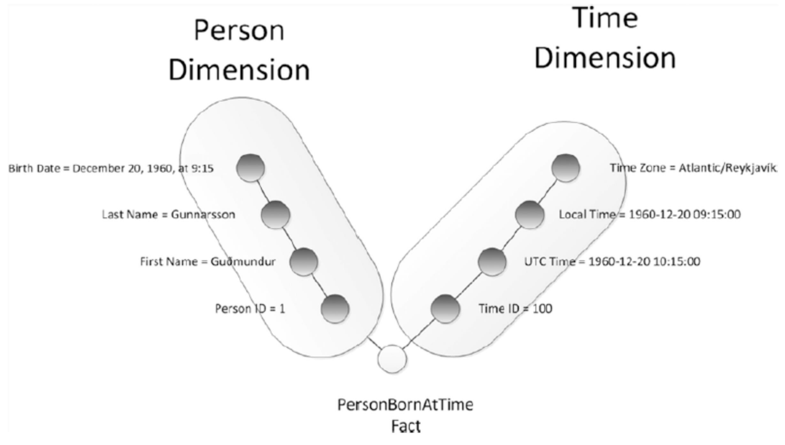
- **Introduction:**
- The Transform superstep allows you, as a data scientist, to take data from the data vault and formulate answers to questions raised by your investigations.
- The transformation step is the data science process that converts results into insights.
- It takes standard data science techniques and methods to attain insight and knowledge about the data that then can be transformed into actionable decisions, which, through storytelling, you can explain to non-data scientists what you have discovered in the data lake.
- The Transform superstep uses the data vault from the process step as its source data.
- The transformations are tuned to work with the five dimensions of the data vault.

- *** Dimension Consolidation:**
- The data vault consists of five categories of data, with linked relationships and additional characteristics in satellite hubs.
- To perform dimension consolidation, you start with a given relationship in the data vault and construct a sun model for that relationship.
- The data warehouse is the only data structure delivered from the Transform step.

- **Sun Model:**
- The use of sun models is a technique that enables the data scientist to perform consistent dimension consolidation, by explaining the intended data relationship with the business, without exposing it to the technical details required to complete the transformation processing.
- **Person-to-Time Sun Model:**

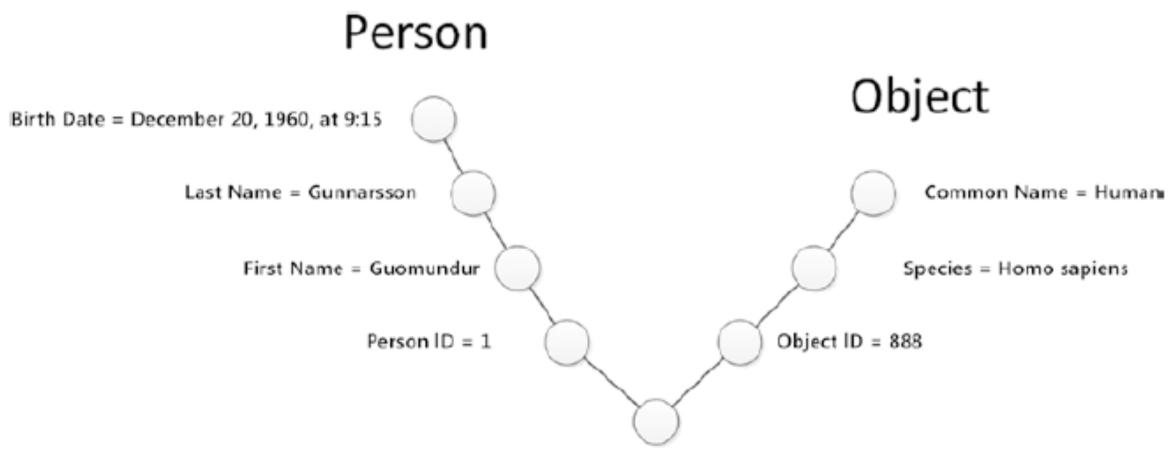

Person
Time
Birth Date = December 20, 1960, at 9:15
Last Name = Gunnarsson
First Name = Guomundur
Person ID = 1
Time Zone = Atlantic/Reykjavik
Local Time = 1960-12-20 09:15:00
UTC Time = 1960-12-20 10:15:00
Time ID = 100

-
- The sun model is constructed to show all the characteristics from the two data vault hub categories you are planning to extract.
- It explains how you will create two dimensions and a fact via the Transform step.

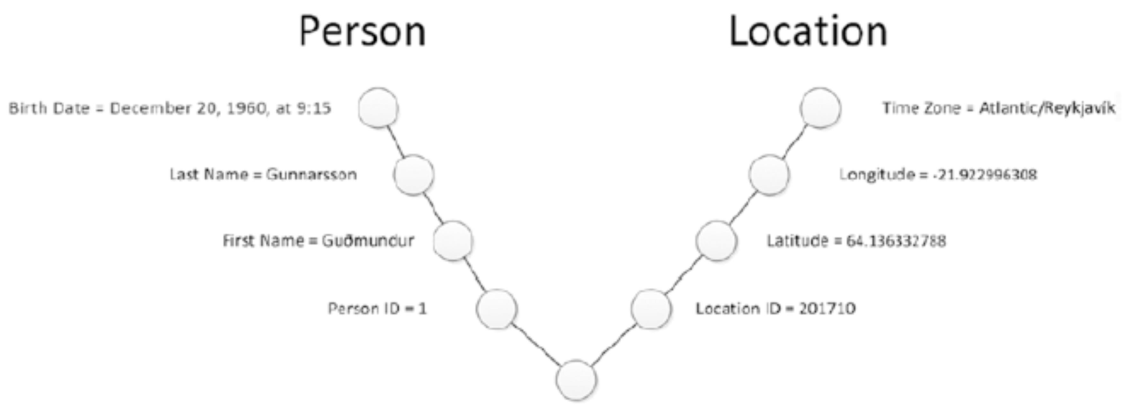

Person Dimension
Time Dimension
Birth Date = December 20, 1960, at 9:15
Last Name = Gunnarsson
First Name = Guðmundur
Person ID = 1
Time Zone = Atlantic/Reykjavik
Local Time = 1960-12-20 09:15:00
UTC Time = 1960-12-20 10:15:00
Time ID = 100
PersonBornAtTime Fact

-

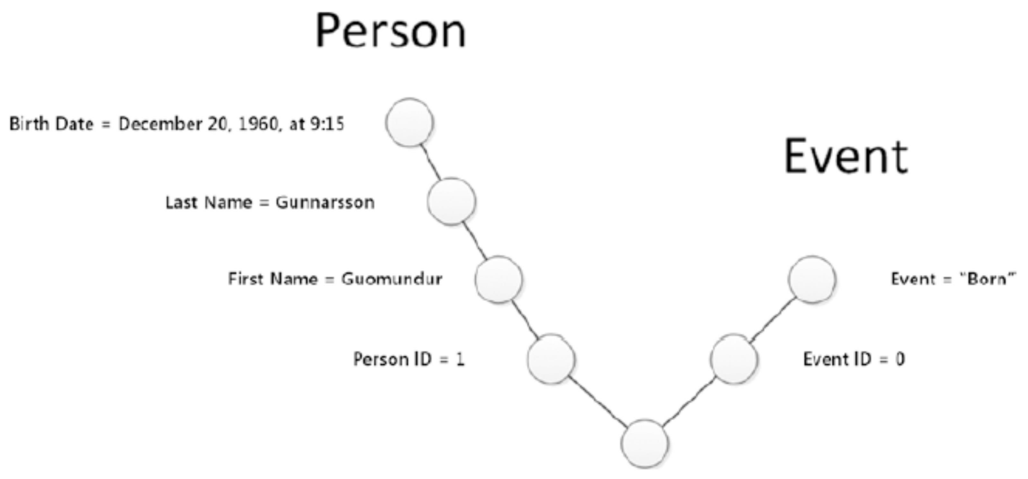
- The sun model explains that Guðmundur Gunnarsson was born on December 20, 1960, at 9:15 in Iceland.
- **2) Person-to-Object Sun Model: (Sun model for the PersonIsSpecies fact)**



-
- **3) Person-to-Location Sun Model: (PersonAtLocation fact)**



-
- **4)Person-to-Event Sun Model: (PersonBorn fact)**



-

- <u>*** Transforming with Data Science:**</u>
- 1) Steps of Data Exploration and Preparation
- You must keep detailed notes of what techniques you employed to prepare the data.
- Make sure you keep your data traceability matrix up to date after each data engineering step has completed.
- Update your Data Lineage and Data Providence, to ensure that you have both the technical and business details for the entire process.

- (standard transform checkpoints)
- **2) Missing Value Treatment:**
- You must describe in detail what the missing value treatments are for the data lake transformation.
- **Why Missing Value Treatment Is Required:**
- Explain with notes on the data traceability matrix why there is missing data in the data lake.
- Remember: Every inconsistency in the data lake is conceivably the missing insight your customer is seeking from you as a data scientist.
- So, find them and explain them.
- **Why Data Has Missing Values:**
- The 5 Whys is the technique that helps you to get to the root cause of your analysis.
- The use of cause-and-effect fishbone diagrams will assist you to resolve those questions.
- common reasons for missing data:
- • Data fields renamed during upgrades
- • Migration processes from old systems to new systems where mappings were incomplete
- • Incorrect tables supplied in loading specifications by subject-matter expert
- • Data simply not recorded, as it was not available
- **What Methods Treat Missing Values?**
- During your progress through the supersteps, you have used many techniques to resolve missing data.
- Record them in your lineage, but also make sure you collect precisely how each technique applies to the processing flow.
-
- **Techniques of Outlier Detection and Treatment:**
- During the processing, you will have detected several outliers that are not complying with your expected ranges, e.g., you expected “Yes” or “No” but found some “N/A”s, or you expected number ranges between 1 and 10 but got 11, 12, and 13 also. These out-of-order items are the outliers.
- **A) Elliptic Envelope:**
- The basic idea is to assume that a data set is from a known distribution and then evaluate any entries not complying to that assumption.
- The scikit-learn package provides an object covariance.EllipticEnvelope that fits a robust covariance estimate to the data, and thus fits an ellipse to the central data points, ignoring points outside the central mode.
- EllipticEnvelope(support_fraction=1., contamination=0.261).
- The support_fraction is the portion of the complete population you want to use to determine the border between inliers and outliers.
- In this case, we use 1, which means 100%.
- The contamination is the indication of what portion of the population could be outliers, hence, the amount of contamination of the data set, i.e., the proportion of outliers in the data set.
- **B) Isolation Forest:**
- One efficient way of performing outlier detection in high-dimensional data sets is to use random forests.
- The ensemble.IsolationForest tool “isolates” observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature.

- Because recursive partitioning can be represented by a tree structure, the number of splittings required to isolate a sample is equivalent to the path length from the root node to the terminating node.
- This path length, averaged over a forest of such random trees, is a measure of normality and our decision function.
- **C) Novelty Detection:**
- Novelty detection simply performs an evaluation in which we add one more observation to a data set.
- Is the new observation so different from the others that we can doubt that it is regular?
- (I.e., does it come from the same distribution?) Or, on the contrary, is it so similar to the other that we cannot distinguish it from the original observations?
- This is the question addressed by the novelty detection tools and methods.
- **D) Local Outlier Factor:**
- An efficient way to perform outlier detection on moderately high-dimensional data sets is to use the local outlier factor (LOF) algorithm.
- The neighbors.LocalOutlierFactor algorithm computes a score (called a local outlier factor) reflecting the degree of abnormality of the observations.
- It measures the local density deviation of a given data point with respect to its neighbors.
- The idea is to detect the samples that have a substantially lower density than their neighbors.
- In practice, the local density is obtained from the k-nearest neighbors.
- The LOF score of an observation is equal to the ratio of the average local density of its k-nearest neighbors and its own local density.
- A normal instance is expected to have a local density like that of its neighbors, while abnormal data are expected to have a much smaller local density.
-
- **2) Common Feature Extraction Techniques:**
- **A) Binning:**
- Binning is a technique that is used to reduce the complexity of data sets, to enable the data scientist to evaluate the data with an organized grouping technique.
- Binning is a good way for you to turn continuous data into a data set that has specific features that you can evaluate for patterns.
- One of the methods is to digitize function.
- import numpy
- data = numpy.random.random(100)
- bins = numpy.linspace(0, 1, 10)
- digitized = numpy.digitize(data, bins)
- bin_means = [data[digitized == i].mean() for i in range(1, len(bins))]
- print(bin_means)
- **B) Averaging:**
- The use of averaging enables you to reduce the amount of records you require to report any activity that demands a more indicative, rather than a precise, total.
- Example:
- Create a model that enables you to calculate the average position for ten sample points.
- First, set up the ecosystem.
- import numpy as np
- import pandas as pd
- Create two series to model the latitude and longitude ranges.
- LatitudeData = pd.Series(np.array(range(-90,91,1)))
- LongitudeData = pd.Series(np.array(range(-180,181,1)))

- You then select 10 samples for each range:
- LatitudeSet=LatitudeData.sample(10)
- LongitudeSet=LongitudeData.sample(10)
- Calculate the average of each.
- LatitudeAverage = np.average(LatitudeSet)
- LongitudeAverage = np.average(LongitudeSet)
- See your results.
- print('Latitude', LatitudeSet , 'Latitude (Avg):', LatitudeAverage)
- print('Longitude', LongitudeSet , 'Longitude (Avg):', LongitudeAverage)
-
- **C) Latent Dirichlet Allocation (LDA):**
- A latent Dirichlet allocation (LDA) is a statistical model that allows sets of observations to be explained by unobserved groups that elucidates why they match or belong together within text documents.
- This technique is useful when investigating text from a collection of documents that are common in the data lake, as companies store all their correspondence in a data lake.
- This model is also useful for Twitter or e-mail analysis.
- Example:
- import numpy as np
- import lda
- import lda.datasets
- X = lda.datasets.load_reuters()
- vocab = lda.datasets.load_reuters_vocab()
- titles = lda.datasets.load_reuters_titles()
- X.shape
- X.sum()

- *** Hypothesis Testing:**
- Hypothesis testing is the process by which statistical tests are used to check if a hypothesis is true, by using data.
- Based on hypothetical testing, data scientists choose to accept or reject the hypothesis.
- When an event occurs, it can be a trend or happen by chance.
- To check whether the event is an important occurrence or just happenstance, hypothesis testing is necessary.
- The following two are most popular tests:
- **A) T-Test:**
- A t-test is a popular statistical test to make inferences about single means or inferences about two means or variances, to check if the two groups' means are statistically different from each other, where n < 30 and standard deviation is unknown.
- Example:
- First you set up the ecosystem, as follows:
- import numpy as np
- from scipy.stats import ttest_ind, ttest_ind_from_stats
- from scipy.special import stdtr
- Create a set of "unknown" data.
- nSet=1
- if nSet==1:
- a = np.random.randn(40)
- b = 4*np.random.randn(50)
- if nSet==2:

- a=np.array([27.1,22.0,20.8,23.4,23.4,23.5,25.8,22.0,24.8,20.2,21.9, 22.1, 22.9, 20.5, 24.4])
- b=np.array([27.1,22.0,20.8,23.4,23.4,23.5,25.8,22.0,24.8,20.2,21.9, 22.1, 22.9, 20.5, 24.41])
- you can create many sets (here upto 5 sets)
- First, you will use scipy's t-test.
- # Use scipy.stats.ttest_ind.
- t, p = ttest_ind(a, b, equal_var=False)
- print("t-Test_ind: t = %g p = %g" % (t, p))
- Second, you will get the descriptive statistics.
- # Compute the descriptive statistics of a and b.
- abar = a.mean()
- avar = a.var(ddof=1)
- na = a.size
- adof = na - 1
- bbar = b.mean()
- bvar = b.var(ddof=1)
- nb = b.size
- bdof = nb – 1
- # Use scipy.stats.ttest_ind_from_stats.
- t2, p2 = ttest_ind_from_stats(abar, np.sqrt(avar), na, bbar, np.sqrt(bvar), nb,
- equal_var=False)
- print("t-Test_ind_from_stats: t = %g p = %g" % (t2, p2))
- Third, you can use the formula to calculate the test.
- # Use the formulas directly.
- tf = (abar - bbar) / np.sqrt(avar/na + bvar/nb)
- dof = (avar/na + bvar/nb)**2 / (avar**2/(na**2*adof) + bvar**2/ (nb**2*bdof))
- pf = 2*stdtr(dof, -np.abs(tf))
- print("Formula: t = %g p = %g" % (tf, pf))
- P=1-p
- if P < 0.001:
- print('Statistically highly significant:',P)
- else:
- if P < 0.05:
- print('Statistically significant:',P)
- else:
- print('No conclusion')

- **2)Chi-Square Test:**
- A chi-square (or squared [χ2]) test is used to examine if two distributions of categorical variables are significantly different from each other.
- Example:
- These are generated with five different data sets.
- First, set up the ecosystem.
- import numpy as np
- import scipy.stats as st
- Create data sets.
- np.random.seed(1)
- # Create sample data sets.
- nSet=1

- if nSet==1:
- a = abs(np.random.randn(50))
- b = abs(50*np.random.randn(50))
- (and so on upto set 5)
- obs = np.array([a,b])
- Perform the test.
- chi2, p, dof, expected = st.chi2_contingency(obs)
- Display the results.
- msg = "Test Statistic : {}\np-value: {}\ndof: {}\n"
- print(msg.format(chi2, p , dof,expected))
- P=1-p
- if P < 0.001:
- print('Statistically highly significant:',P)
- else:
- if P < 0.05:
- print('Statistically significant:',P)
- else:
- print('No conclusion')

- ***Overfitting and Underfitting:**
- Overfitting and underfitting are major problems when data scientists retrieve data insights from the data sets they are investigating.
- Overfitting is when the data scientist generates a model to fit a training set perfectly, but it does not generalize well against an unknown future real-world data set, as the data science is so tightly modeled against the known data set, the most minor outlier simply does not get classified correctly.
- The solution only works for the specific data set and no other data set.
- Underfitting the data scientist's results into the data insights has been so nonspecific that to some extent predictive models are inappropriately applied or questionable as regards to insights.
- Your data science must offer a significant level of insight for you to secure the trust of your customers, so they can confidently take business decisions, based on the insights you provide them.
-
- **A) Polynomial Features:**
- The polynomic formula is the following:
- $(a_1x+b_1)(a_2x+b_2)=a_1a_2x^2+(a_1b_2+a_2b_1)x+b_1b_2.$
- The polynomial feature extraction can use a chain of polynomic formulas to create a hyperplane that will subdivide any data sets into the correct cluster groups.
- The higher the polynomic complexity, the more precise the result that can be achieved.
- Example:
- import numpy as np
- import matplotlib.pyplot as plt
- from sklearn.linear_model import Ridge
- from sklearn.preprocessing import PolynomialFeatures
- from sklearn.pipeline import make_pipeline
- def f(x):
- """ function to approximate by polynomial interpolation"""
- return x * np.sin(x)

- # generate points used to plot
- x_plot = np.linspace(0, 10, 100)
- # generate points and keep a subset of them
- x = np.linspace(0, 10, 100)
- rng = np.random.RandomState(0)
- rng.shuffle(x)
- x = np.sort(x[:20])
- y = f(x)
- # create matrix versions of these arrays
- X = x[:, np.newaxis]
- X_plot = x_plot[:, np.newaxis]
- colors = ['teal', 'yellowgreen', 'gold']
- lw = 2
- plt.plot(x_plot, f(x_plot), color='cornflowerblue', linewidth=lw, label="Ground Truth")
- plt.scatter(x, y, color='navy', s=30, marker='o', label="training points")
- for count, degree in enumerate([3, 4, 5]):
- model = make_pipeline(PolynomialFeatures(degree), Ridge())
- model.fit(X, y)
- y_plot = model.predict(X_plot)
- plt.plot(x_plot, y_plot, color=colors[count], linewidth=lw, label="Degree %d" % degree)
- plt.legend(loc='lower left')
- plt.show()

- **B) Common Data-Fitting Issue:**
- These higher order polynomic formulas are, however, more prone to overfitting, while lower order formulas are more likely to underfit.
- It is a delicate balance between two extremes that support good data science.
- Example:
- import numpy as np
- import matplotlib.pyplot as plt
- from sklearn.pipeline import Pipeline
- from sklearn.preprocessing import PolynomialFeatures
- from sklearn.linear_model import LinearRegression
- from sklearn.model_selection import cross_val_score
- def true_fun(X):
- return np.cos(1.5 * np.pi * X)
- np.random.seed(0)
- n_samples = 30
- degrees = [1, 4, 15]
- X = np.sort(np.random.rand(n_samples))
- y = true_fun(X) + np.random.randn(n_samples) * 0.1
- plt.figure(figsize=(14, 5))
- for i in range(len(degrees)):
- ax = plt.subplot(1, len(degrees), i + 1)
- plt.setp(ax, xticks=(), yticks=())
- polynomial_features = PolynomialFeatures(degree=degrees[i],
- include_bias=False)
- linear_regression = LinearRegression()
- pipeline = Pipeline([("polynomial_features", polynomial_features),
- ("linear_regression", linear_regression)])

- pipeline.fit(X[:, np.newaxis], y)
- # Evaluate the models using crossvalidation
- scores = cross_val_score(pipeline, X[:, np.newaxis], y,
- scoring="neg_mean_squared_error", cv=10)
- X_test = np.linspace(0, 1, 100)
- plt.plot(X_test, pipeline.predict(X_test[:, np.newaxis]), label="Model")
- plt.plot(X_test, true_fun(X_test), label="True function")
- plt.scatter(X, y, edgecolor='b', s=20, label="Samples")
- plt.xlabel("x")
- plt.ylabel("y")
- plt.xlim((0, 1))
- plt.ylim((-2, 2))
- plt.legend(loc="best")
- plt.title("Degree {}\nMSE = {:.2e}(+/- {:.2e})".format(
- degrees[i], -scores.mean(), scores.std()))
- plt.show()

- *** Precision-Recall:**
- Precision-recall is a useful measure for successfully predicting when classes are extremely imbalanced.
- In information retrieval,
- • Precision is a measure of result relevancy.
- • Recall is a measure of how many truly relevant results are returned.
- **A) Precision-Recall Curve:**
- The precision-recall curve shows the trade-off between precision and recall for different thresholds.
- A high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate. High scores for both shows that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall).
- A system with high recall but low precision returns many results, but most of its predicted labels are incorrect when compared to the training labels.
- A system with high precision but low recall is just the opposite, returning very few results, but most of its predicted labels are correct when compared to the training labels.
- An ideal system with high precision and high recall will return many results, with all results labeled correctly.
- Precision (P) is defined as the number of true positives (Tp) over the number of true positives (Tp) plus the number of false positives (Fp).

$$P = \frac{\mathrm{Tp}}{Tp + Fp}$$

-
- Recall (R) is defined as the number of true positives (Tp) over the number of true positives (Tp) plus the number of false negatives (Fn).

$$R = \frac{\mathrm{Tp}}{Tp + Fn}$$

-
- Accuracy (A) is defined as

- $$A = \frac{Tp + Tn}{Tp + Fp + Tn + Fn}$$
-
- **B) Sensitivity and Specificity:**
- Sensitivity and specificity are statistical measures of the performance of a binary classification test, also known in statistics as a classification function.
- Sensitivity (also called the true positive rate, the recall, or probability of detection) measures the proportion of positives that are correctly identified as such (e.g., the percentage of sick people who are correctly identified as having the condition).
- Specificity (also called the true negative rate) measures the proportion of negatives that are correctly identified as such (e.g., the percentage of healthy people who are correctly identified as not having the condition).
-
- **C) F1-Measure:**
- The F1-score is a measure that combines precision and recall in the harmonic mean of precision and recall.
  $$F1 = 2 * \frac{P * R}{P + R}$$
-
- Example:
- Create simple data
- from sklearn import svm, datasets
- from sklearn.model_selection import train_test_split
- import numpy as np
- iris = datasets.load_iris()
- X = iris.data
- y = iris.target
- # Add noisy features
- random_state = np.random.RandomState(0)
- n_samples, n_features = X.shape
- X = np.c_[X, random_state.randn(n_samples, 200 * n_features)]
- # Limit to the two first classes, and split into training and test
- X_train, X_test, y_train, y_test = train_test_split(X[y < 2], y[y < 2],
- test_size=.5, random_state=random_state)
- # Create a simple classifier
- classifier = svm.LinearSVC(random_state=random_state)
- classifier.fit(X_train, y_train)
- y_score = classifier.decision_function(X_test)
- Compute the average precision score
- from sklearn.metrics import average_precision_score
- average_precision = average_precision_score(y_test, y_score)
- print('Average precision-recall score: {0:0.2f}'.format(average_precision))
-
- **D) Receiver Operating Characteristic (ROC) Analysis Curves:**
- A receiver operating characteristic (ROC) analysis curve is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.
- The ROC curve plots the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The true positive rate is also known as sensitivity, recall, or probability of detection.

- You will find the ROC analysis curves useful for evaluating whether your classification or feature engineering is good enough to determine the value of the insights you are finding.
- This helps with repeatable results against a real-world data set.
- So, if you suggest that your customers should take a specific action as a result of your findings, ROC analysis curves will support your advice and insights but also relay the quality of the insights at given parameters.
- import numpy as np
- from scipy import interp
- import matplotlib.pyplot as plt
- from sklearn import svm, datasets
- from sklearn.metrics import roc_curve, auc
- from sklearn.model_selection import StratifiedKFold
- # Data IO and generation
- # Import some data to play with
- iris = datasets.load_iris()
- X = iris.data
- y = iris.target
- X, y = X[y != 2], y[y != 2]
- n_samples, n_features = X.shape
- # Add noisy features
- random_state = np.random.RandomState(0)
- X = np.c_[X, random_state.randn(n_samples, 200 * n_features)]
- # Classification and ROC analysis
- # Run classifier with cross-validation and plot ROC curves
- cv = StratifiedKFold(n_splits=6)
- classifier = svm.SVC(kernel='linear', probability=True,
- random_state=random_state)
- tprs = []
- aucs = []
- mean_fpr = np.linspace(0, 1, 100)
- i = 0
- for train, test in cv.split(X, y):
- probas_ = classifier.fit(X[train], y[train]).predict_proba(X[test])
- # Compute ROC curve and area the curve
- fpr, tpr, thresholds = roc_curve(y[test], probas_[:, 1])
- tprs.append(interp(mean_fpr, fpr, tpr))
- tprs[-1][0] = 0.0
- roc_auc = auc(fpr, tpr)
- aucs.append(roc_auc)
- plt.plot(fpr, tpr, lw=1, alpha=0.3,
- label='ROC fold %d (AUC = %0.2f)' % (i, roc_auc))
- i += 1
- plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
- label='Luck', alpha=.8)
- mean_tpr = np.mean(tprs, axis=0)
- mean_tpr[-1] = 1.0
- mean_auc = auc(mean_fpr, mean_tpr)
- std_auc = np.std(aucs)
- plt.plot(mean_fpr, mean_tpr, color='b',
- label=r'Mean ROC (AUC = %0.2f $\pm$ %0.2f)' % (mean_auc, std_auc), lw=2, alpha=.8)

- std_tpr = np.std(tprs, axis=0)
- tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
- tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
- plt.fill_between(mean_fpr, tprs_lower, tprs_upper, color='gray', alpha=.2, label=r'$\pm$ 1 std. dev.')
- plt.xlim([-0.05, 1.05])
- plt.ylim([-0.05, 1.05])
- plt.xlabel('False Positive Rate')
- plt.ylabel('True Positive Rate')
- plt.title('Receiver operating characteristic example')
- plt.legend(loc="lower right")
- plt.show()

- ***Cross-Validation Test:**
- Cross-validation is a model validation technique for evaluating how the results of a statistical analysis will generalize to an independent data set.
- It is mostly used in settings where the goal is the prediction.
- Knowing how to calculate a test such as this enables you to validate the application of your model on real-world, i.e., independent data sets.
- import numpy as np
- from sklearn.model_selection import cross_val_score
- from sklearn import datasets, svm
- import matplotlib.pyplot as plt
- digits = datasets.load_digits()
- X = digits.data
- y = digits.target
- Let's pick three different kernels and compare how they will perform.
- kernels=['linear', 'poly', 'rbf']
- for kernel in kernels:
- svc = svm.SVC(kernel=kernel)
- C_s = np.logspace(-15, 0, 15)
- scores = list()
- scores_std = list()
- for C in C_s:
- svc.C = C
- this_scores = cross_val_score(svc, X, y, n_jobs=1)
- scores.append(np.mean(this_scores))
- scores_std.append(np.std(this_scores))