The Best of Both Worlds: Analytics with SAS(R) Viya(R) and Python

- Understand the massively parallel processing capabilities in SAS Viya
- Integrate Python and SAS Viya to process data throughout the analytics life cycle
- Create a dashboard using SAS Visual Analytics

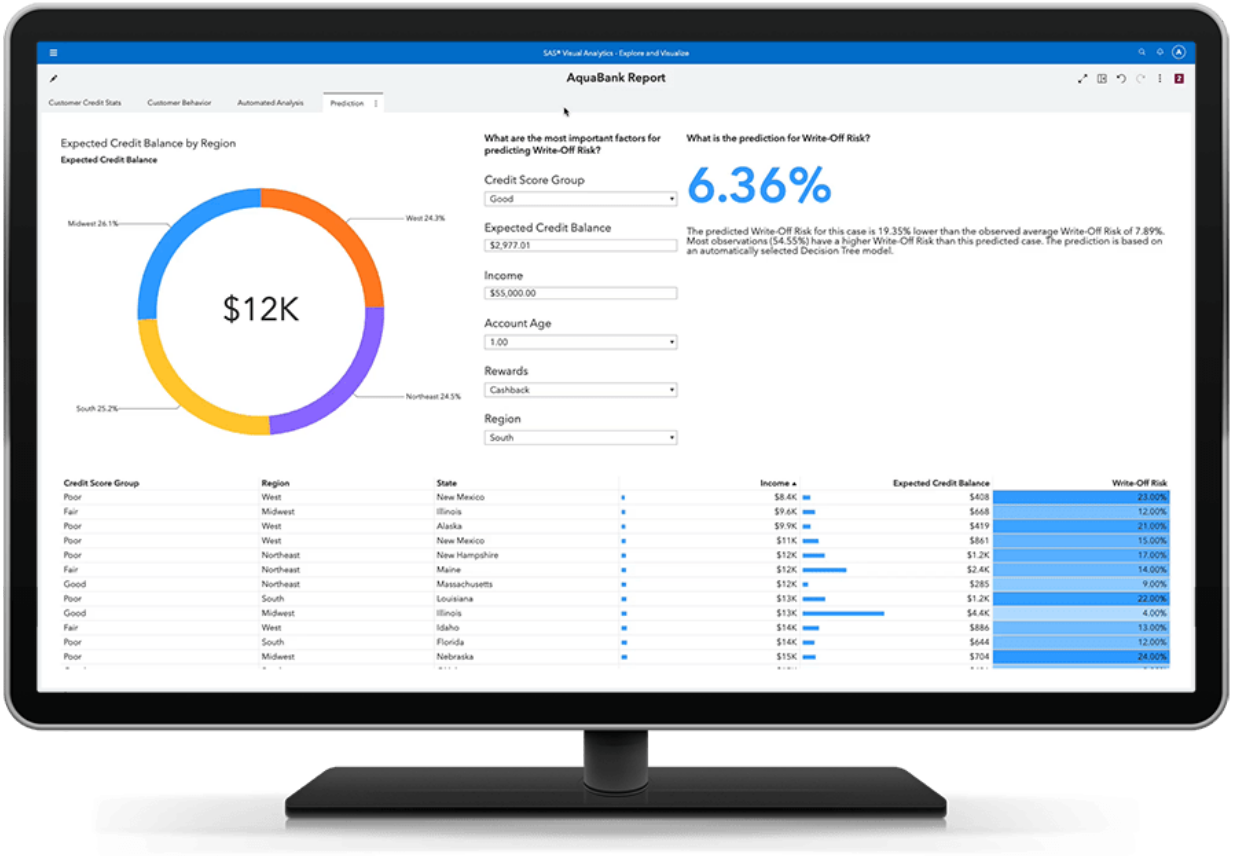
Project Goal

1. Prepare the data using the Python SWAT package to process data in a distributed environment.
2. Use SAS Visual Analytics to create a dashboard.
3. Learn to create models.

In [2]:
```
##
## Display image function
##

from IPython.display import Image
def display_image(img):
    ''' Display images for presentation'''
    return Image(url=r'https://raw.githubusercontent.com/pestyld/Python-Integration-to
```

In [3]:
```
display_image('00_VA_Basic.png')
```

Out[3]:

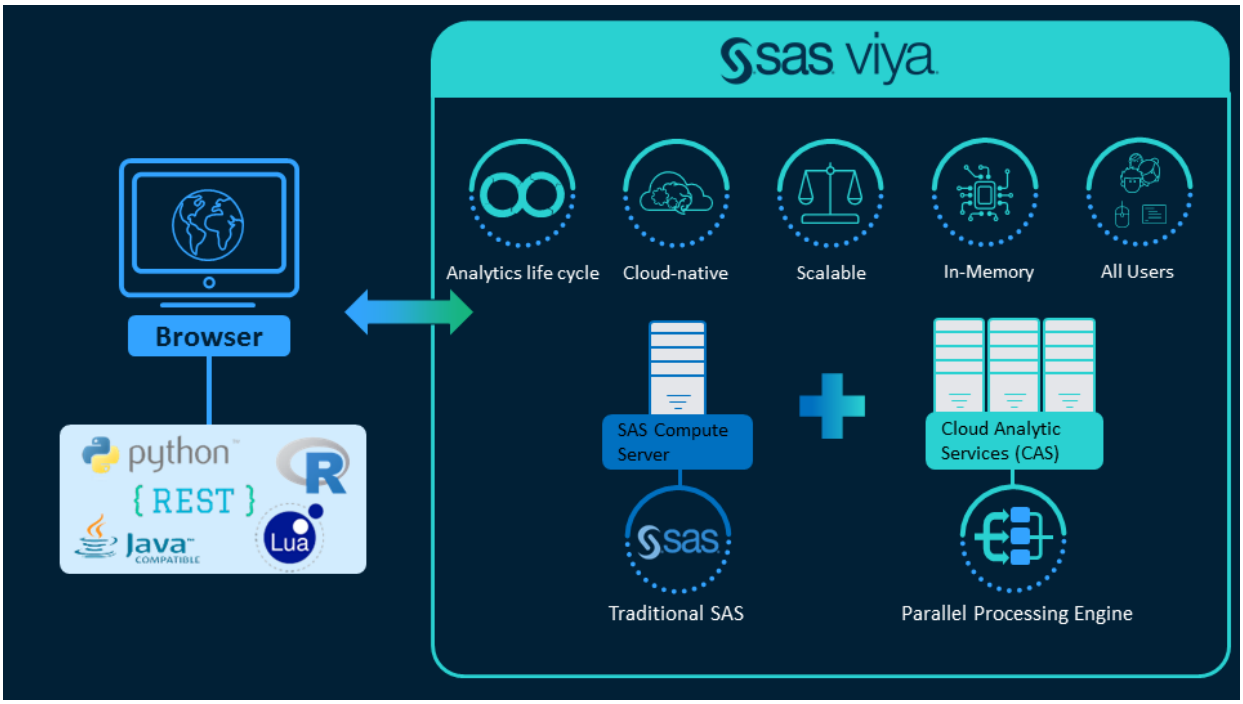


Introduction to SAS Viya

SAS Viya Overview

In [4]:
```
display_image('01_Viya_Overview.png')
```

Out[4]:



Expand notes below each slide for more information

SAS Viya contains a variety of applications to process your data through the entirety of the analytics life cycle, but SAS Viya is much more than that. It's also cloud native, allowing scalable, web-based access to quick, accurate, and reliable analytical insights. Its in-memory engine and parallel processing capabilities can significantly improve the execution speed of your analytics. In its entirety, SAS Viya is a collaborative environment for all users. It enables everyone – data scientists, business analysts, developers, report viewers and executives alike – to collaborate, scale, and operationalize insights.

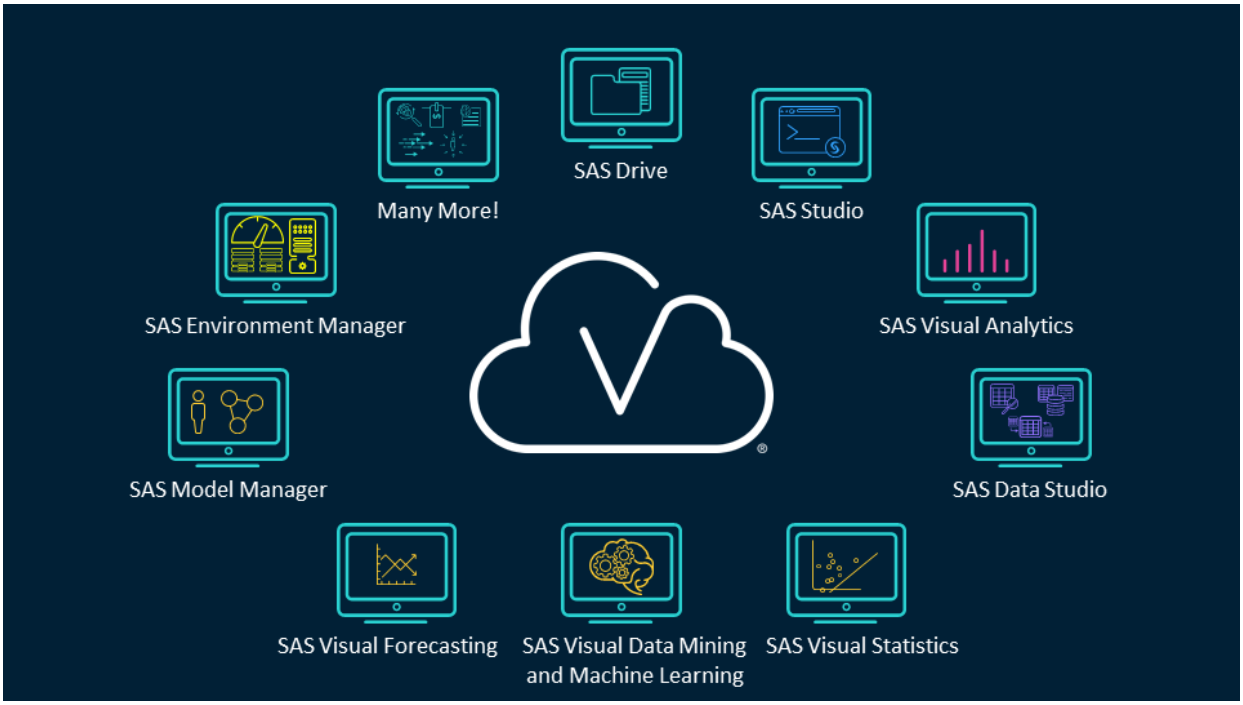
How does all of this happen? SAS Viya includes multiple servers. The SAS Compute Server and SAS Cloud Analytic Services, or the CAS server for short. The SAS Compute Server enables you to execute traditional SAS code. The SAS Compute Server is the Viya equivalent of the SAS®9 workspace server.

Then there's the CAS server. CAS has a massively parallel processing (MPP) architecture that is appropriate for analyzing big data and resource-intensive programs. It performs parallel processing on resident in-memory data to boost performance and produce lightning-fast results. It is optimal for big data and resource-intensive programs like machine learning. In this course, we focus on working with the CAS server in SAS Viya.

All of these features are accessed from a web browser, and SAS Viya provides integration with open-source languages like Python, R, REST, JAVA, and LUA.

In [5]:
```
display_image('02_Viya_Apps.png')
```

Out[5]:



SAS Viya consists of many applications that enable you to work with your data no matter your job role or skill level. This can be done through the collection of integrated AI, analytic, and data management solutions in SAS Viya.

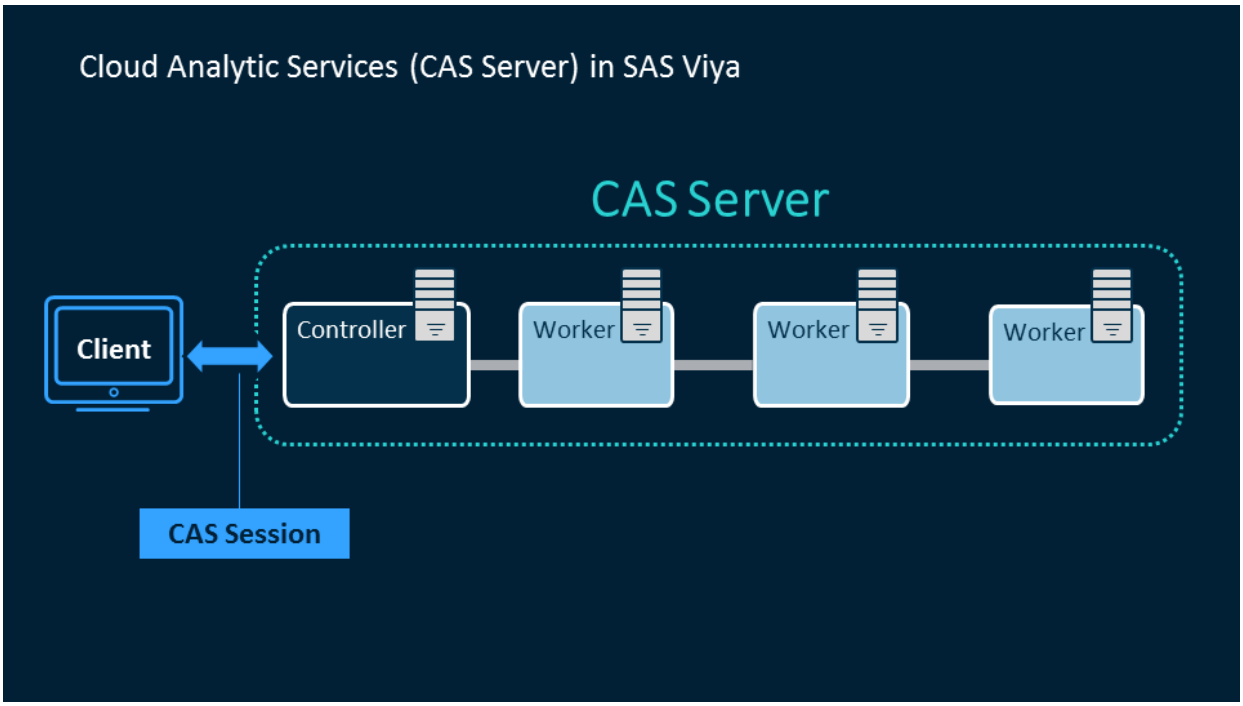
- The entry point for SAS Viya is **SAS Drive**, a collaborative interface for accessing, organizing, and sharing content. This application is the gateway to all other applications in SAS Viya.
- **SAS Studio** is the interactive development environment (IDE) in SAS Viya that enables you to program, build flows, and even use point-and-click tools to process your data.
- **SAS Visual Analytics** enables you to visualize data, build statistical models, and create interactive dashboards with an easy-to-use point-and-click interface.
- **SAS Data Studio** is the data preparation application that enables you to easily prepare data using point-and-click transforms. It also enables you to easily embed custom code if necessary.
- **SAS Visual Statistics**, **SAS Visual Data Mining and Machine Learning**, and **SAS Visual Forecasting** are applications built for data scientists. Users can interactively create and refine predictive machine learning models and forecasts.
- **SAS Model Manager** streamlines the model life cycle
- **SAS Environment Manager** manages the entire environment.

These are just a few of the many applications available in SAS Viya. All of these applications enable individuals in an organization to work with data.

CAS Server Overview

In [6]:
```
display_image('03_CAS_Overview1.png')
```

Out[6]:



The CAS server is the cloud-native, high-performance in-memory analytics and massively parallel processing engine. The CAS server is configured to run on multiple machines. Typically, you will have one controller node and several worker nodes. You need to make a connection to the CAS server from your client. This is called a CAS session or CAS connection.

SAS Documentation - SAS® Cloud Analytic Services: Fundamentals

For data to be processed in CAS, data is copied into memory from some data source, and data blocks are automatically distributed across the worker nodes and loaded into memory. This is called serial loading. Once data is loaded into memory it persists in memory until explicitly dropped or the CAS session ends.

You can also load data into memory in parallel. View the following resources for additional information:

- Five Approaches for High-Performance Data Loading to the SAS® Cloud Analytic Services Server -Parallel data load to SAS Viya
- 4 Approaches for Parallel Data Loading to CAS]
- How to Parallel Load and Compress a SAS® Cloud Analytic Services (CAS) Table

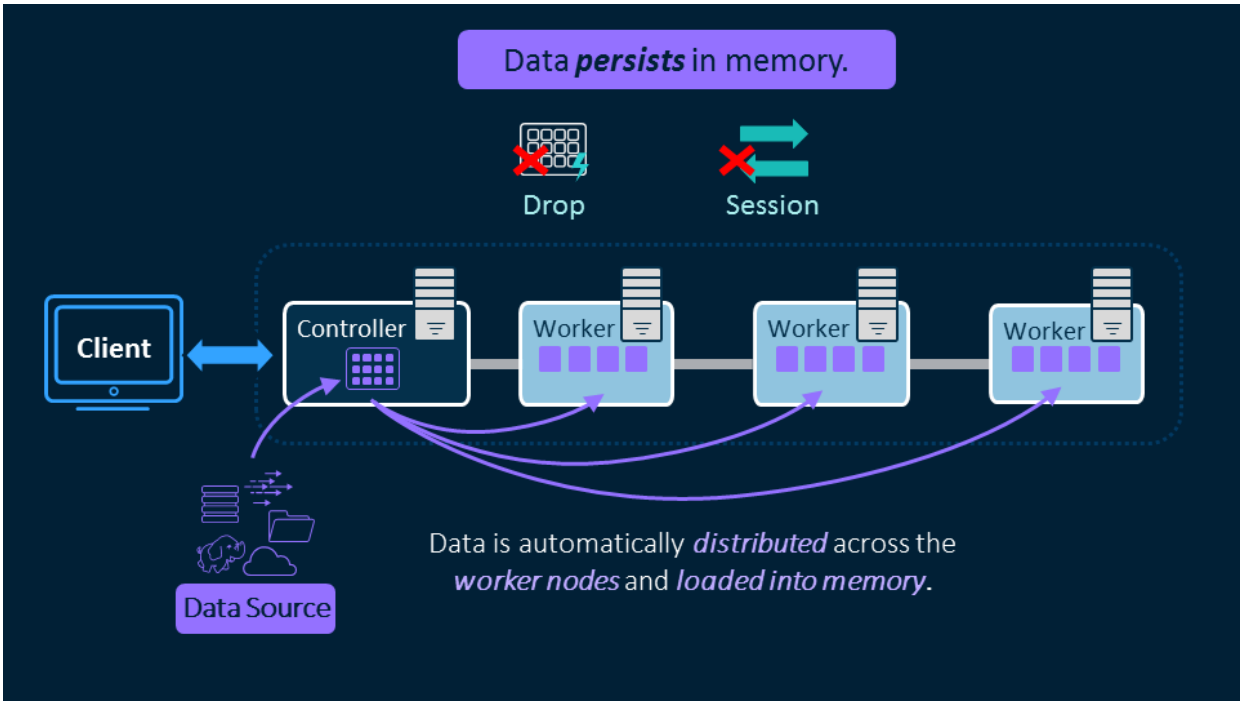
If you persist the data in memory, it enables multiple users to access the same in-memory table without any additional I/O (data load from disk to memory). Because I/O is more resource intensive, having data preloaded into memory speeds up the processing for all users.

For example, person A can execute a variety of Python programs, person B can create dashboards using SAS Visual Analytics, and person C can execute machine learning models. All users can do this using the same in-memory table copy.

In all instances, the controller accepts the programs from the client and distributes code to the workers. The workers perform coordinated parallel processing on their portion of the data, with multiple nodes executing the same actions on different parts of the data at the same time. The controller monitors progress and coordinates re-assembling the result segments produced by each worker. If requested by the client, the controller then returns the results to the client for further processing. Modifications to the in-memory table are not reflected in the physically stored data source. If you want to update the physical data source, the in-memory table must be explicitly saved back to the physical storage location.

In [7]:
```
display_image('03_CAS_Overview2.png')
```

Out[7]:



The CAS server needs to access data from some physical data source. CAS can access data from a variety of sources using SAS Viya Data Connectors, including databases, Hadoop, streaming data, path-based files, and data stored in the cloud.

Once data is loaded into memory on the CAS server, you can process that data using CAS actions. Actions are optimized units of work built for the distributed CAS server. CAS actions provide a range of functionality from managing, processing, analyzing, and modeling data, to executing most traditional SAS DATA step, FedSQL, and even DS2!

CAS actions are organized into groups called CAS action sets, and the actions within a set perform related tasks. You can think of action sets as a package, and the actions within an action set as methods.

For example, the table action set provides many actions for accessing and managing data. These include the caslibInfo action to view available data sources, the tableInfo action to view available tables, the fileInfo action to list the data source files, and the fetch action to retrieve n rows of a table.

You request a specific action by specifying the action set name, a period, and the action name. Specifying the action set name is usually optional. However, there are a few actions with the same name in different action sets. In those cases, the action set name is required. In this course, we typically specify only the action.

CAS actions provide a variety of benefits.

- First, CAS actions are optimized to run in the CAS server's massively parallel processing engine for lightening-fast results.
- Second, the same CAS action is used regardless of programming language or application. That means you can use the same CAS action in CASL, Python, R, and more. This enables

you to easily pass information between languages.

- When using actions in different languages, the results are equivalent (given seeds).
- Lastly, actions are multi-purpose. Actions perform a specific task, and within that task, an action can also perform a variety of other tasks like create a new table, create calculated columns, filter rows and columns, and group data.

The question is, how can you execute actions?

SAS Viya provides a variety of interfaces for executing CAS actions. One way to execute actions on the CAS server is to use the native CAS language, or CASL for short. CASL supports executing CAS actions on the CAS server and uses normal programing logic. If you have experience using languages like Lua, Python, or R, CASL will be an easy language to learn.

Instead of using CASL, you can also execute CAS actions through the CAS API using traditional SAS programming, FedSQL, Java, Python, R, Lua, and REST API. This course focuses on using the Python language.

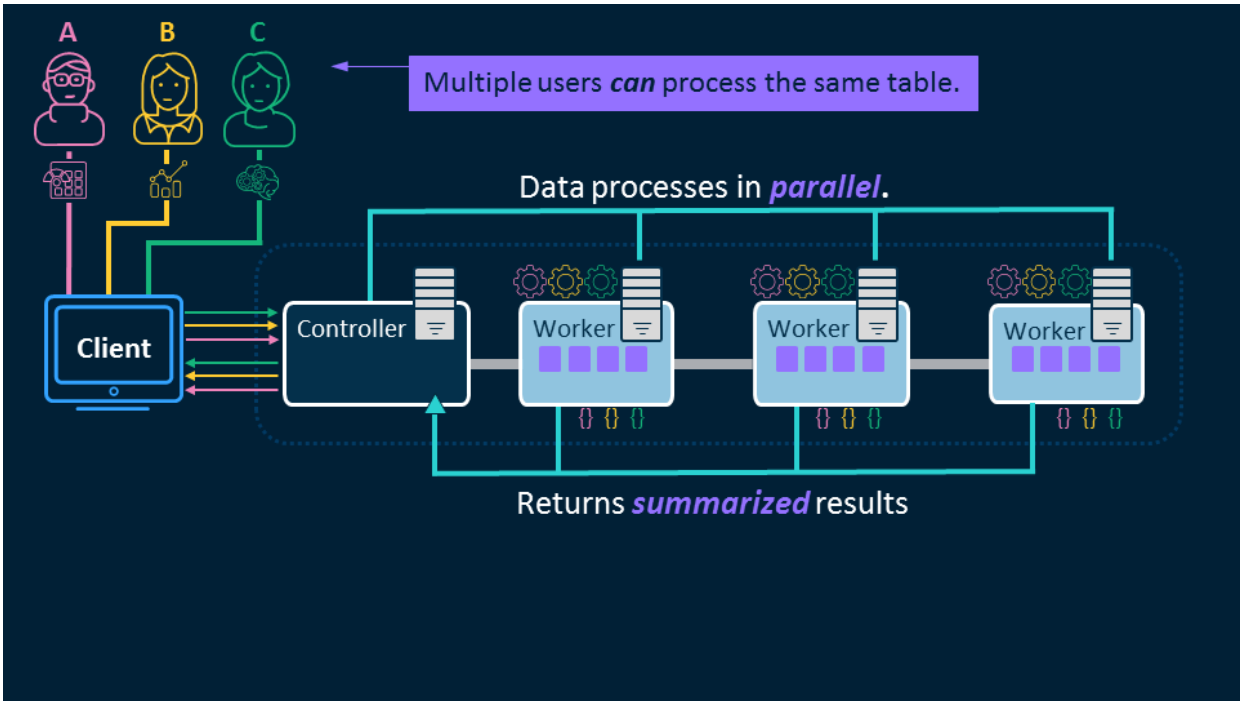
Lastly, you can use a variety of SAS Viya applications like SAS Visual Analytics to create dashboards, SAS Visual Statistics and SAS Visual Data Mining and Machine Learning for modeling and statistics, and a variety of others. All applications execute actions behind the scenes.

NOTES:

1. With the SAS programming language, many PROCs and much of the traditional DATA step is CAS enabled and can be executed on CAS tables. For more information about how to execute traditional SAS code in SAS Viya, you can view the Programming for SAS Viya course: https://support.sas.com/edu/schedules.html?crs=PGVIYA&ctry=US.
2. For Java, you must use the CASClient class.
3. For Python, R, and Luna, the SWAT package is required. Many familiar methods are available through the SWAT package.
4. The CAS API converts the native language syntax to CAS actions behind the scenes.
5. Visit the SAS website for detailed information about all available SAS Viya offerings.

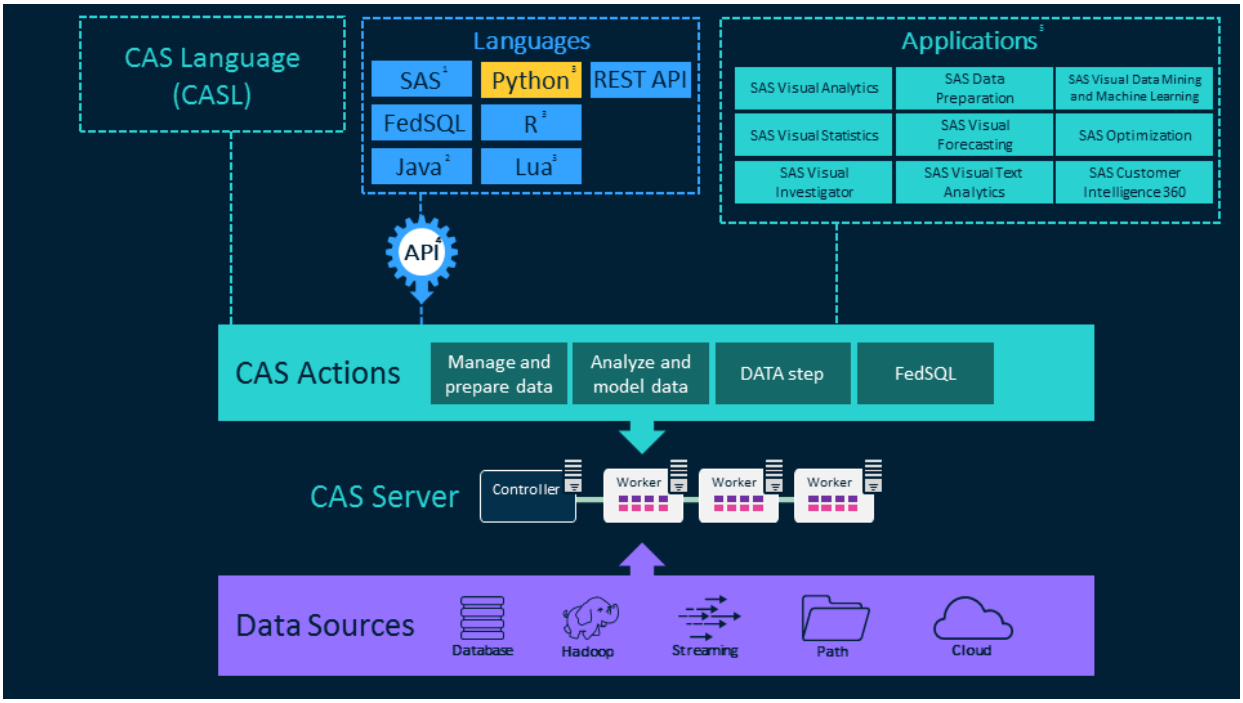
In [8]:
```
display_image('03_CAS_Overview3.png')
```

Out[8]:



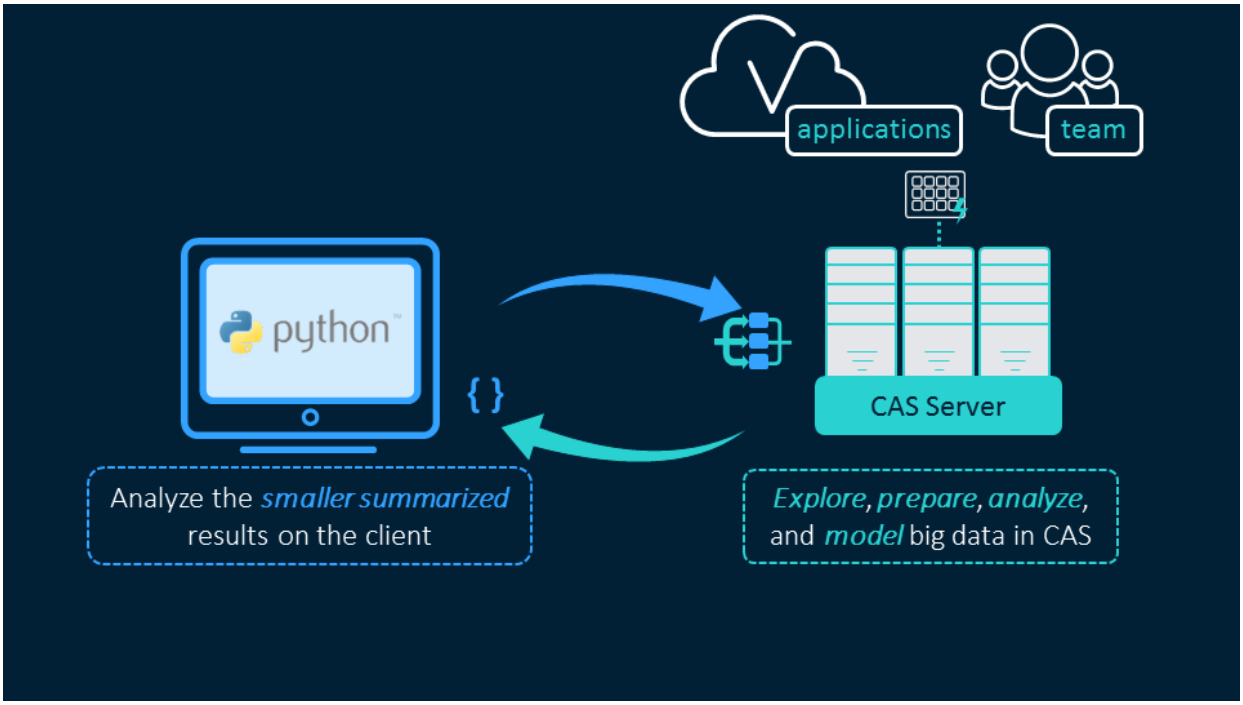
In [9]:
```
display_image('03_CAS_Overview4.png')
```

Out[9]:



In [10]:
```
display_image('05_Client_Server.png')
```

Out[10]:



You can use Python on your client to explore, prepare, analyze, and model big data on the CAS server using familiar Python syntax along with CAS actions to take advantage of the parallel processing power of CAS. If you are preparing data in CAS, you can then share that data with other users or other SAS Viya applications like SAS Visual Analytics to create shareable dashboards.

If you are exploring, analyzing, or modeling data in CAS to take advantage of the in-memory parallel processing power, CAS returns smaller summarized results back to your local Python client. Once the summarized results from the CAS server are returned to your client, you can use native Python packages on the smaller, more manageable data.

In [11]:
```
display_image('07_pandas.png')
```

Out[11]:

As an experienced Python programmer, you most likely are familiar with Python packages like Pandas, Matplotlib, NumPy, and seaborn. You might be thinking, "Great, I can use these packages on data in CAS!"

False!

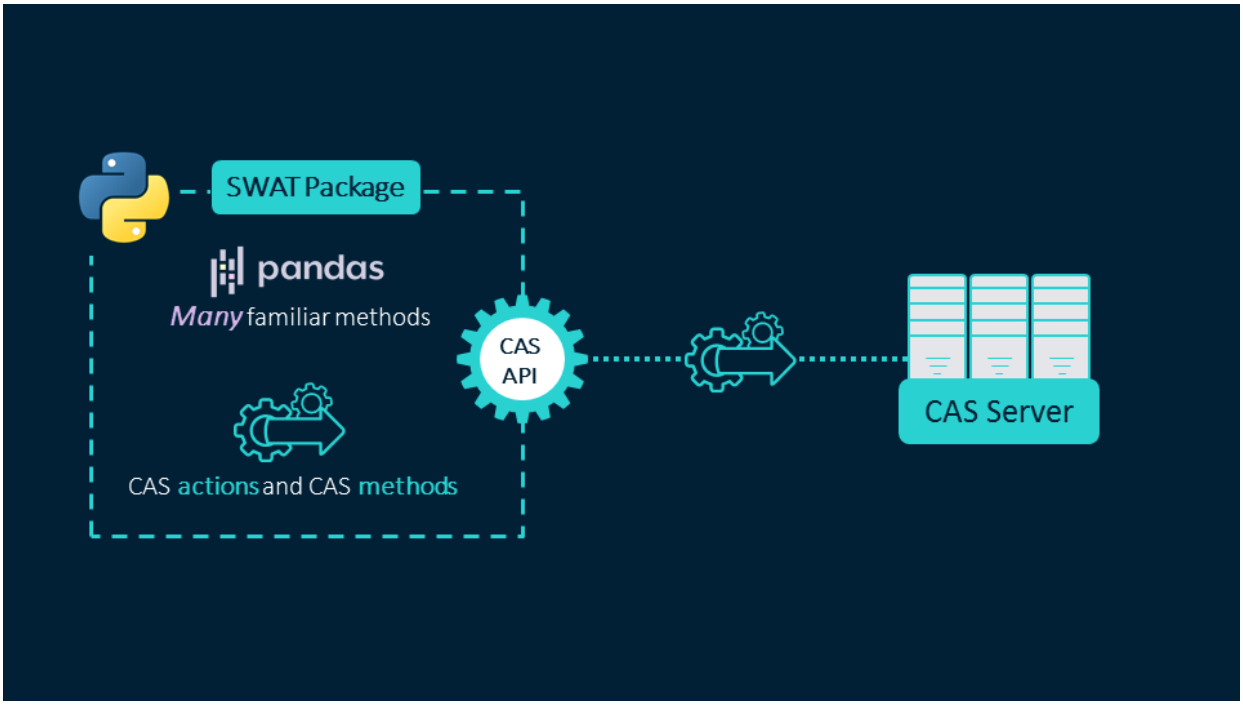
Although these are great packages, they do not execute on data on the CAS server. I'll repeat this because it's very important and a common misconception. Native Python packages do not execute on data on the CAS server.

SWAT Package Overview

SWAT Documentation

In [12]:
```
display_image('08a_swat.png')
```

Out[12]:



In order to process data with Python on the CAS server, you must use the SAS Scripting Wrapper for Analytics Transfer, or the SWAT package for short. This is an open-source package used with Python, R, and Lua. With the SWAT package, you can write a program that connects to the CAS server, quickly load large amounts of data into memory, analyze the data, and then work with the results of your analysis using familiar techniques in the open-source language of your choice.

The Python SWAT package contains many familiar Pandas methods, functions, and attributes that work like their counterparts. However, not all methods and functions are available, so be sure to download the latest version of the SWAT package for the most up-to-date functionality.

The SWAT package also contains an enormous number of CAS actions and CAS methods that can be used to process your data.

For example, with the SWAT package, you can use the familiar head method from the Pandas API on a CAS table to return the first n rows of a CAS table.

In all cases, the code is translated through the CAS API into the same fetch CAS action behind the scenes and sent to the CAS server. The action is processed in CAS, and the results are sent from the CAS server to the client.The CAS API converts familiar Pandas methods into actions behind the scenes, but you can also execute actions directly through the SWAT package.

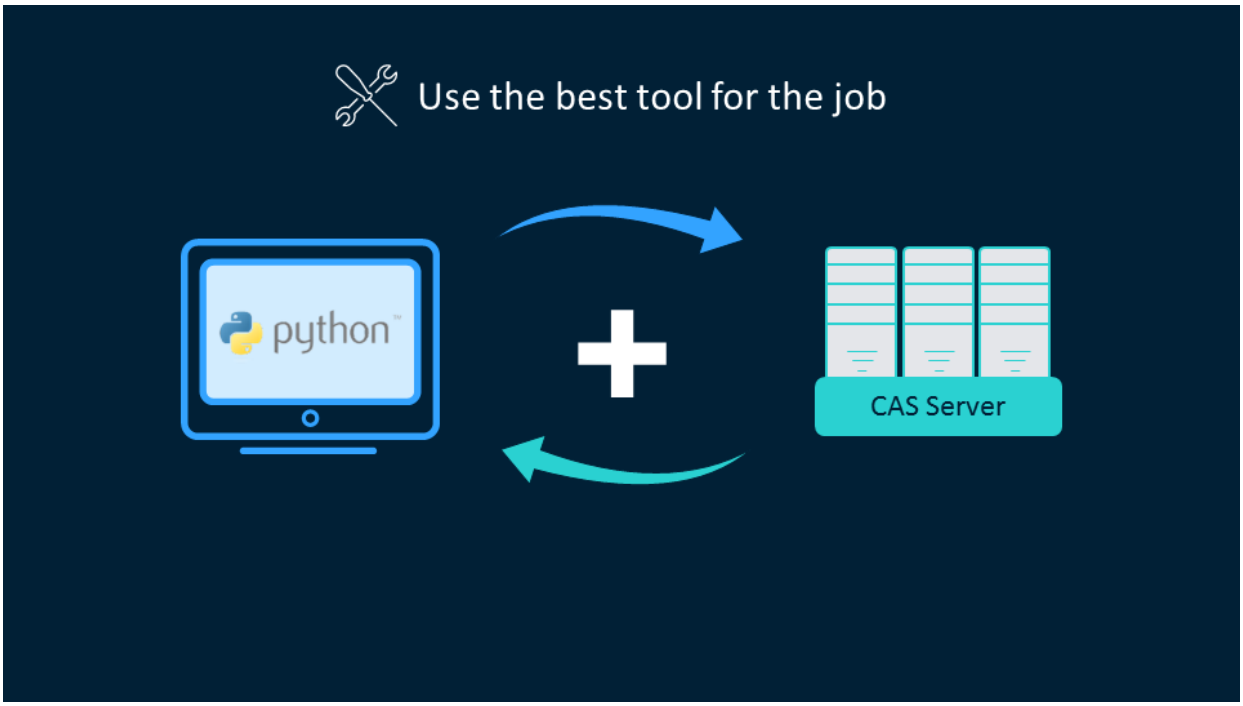
In this example, you can execute the same fetch CAS action in Python. The action is sent to the CAS server, and CAS processes the action and returns the results to the client. The same action can be used in all languages that work with CAS.

The CAS API converts familiar Pandas methods into actions behind the scenes, but you can also execute actions directly through the SWAT package.

In this example, you can execute the same fetch CAS action in Python. The action is sent to the CAS server, and CAS processes the action and returns the results to the client. The same action can be used in all languages that work with CAS.

In [13]:
```
display_image('09_client_server.png')
```

Out[13]:



In the end, having both your Python client and the CAS server offers you a lot of flexibility. The goal is to look at each as a tool and determine the best tool for the job.

Accessing, exploring, preparing and analyzing data using the Python SWAT Package

1. Setup

a. Import packages

In [14]:
```
import sys
import os
import swat
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

## Set pandas column options
pd.set_option('max_colwidth', 200)
pd.set_option('display.max_columns', None)
```

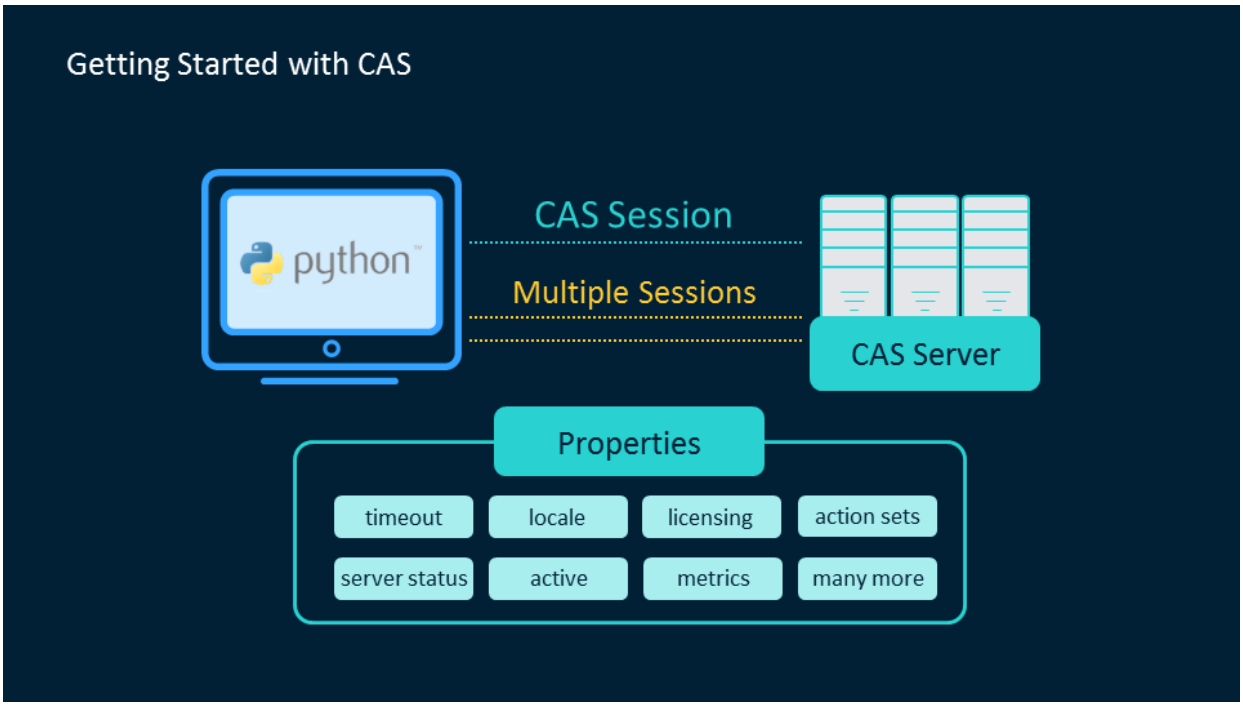
```
## My personal module to connect to CAS. Will not work in other environments. The pack
try:
    from casauth import CASAuth
    print('Imported personal custom CAS auth package')
except:
    print('casauth package not available')


## Check Python and package versions
print(f'Python version:{sys.version.split(" ")[0]}')
print(f'numpy:{np.__version__}')
print(f'pandas:{pd.__version__}')
print(f'swat:{swat.__version__}')
```

```
Imported personal custom CAS auth package
Python version:3.8.16
numpy:1.24.3
pandas:1.5.3
swat:1.13.1
```

b. Connect to the CAS server

In [15]:
```
display_image('10_CAS_session.png')
```

Out[15]:



To begin working with CAS, you need to make a connection to the CAS server. This is known as a CAS session. A CAS session contains a variety of information and properties. These include server status and licensing, as well as session properties like timeout, locale, active caslib, metrics, and more.

You can also create multiple CAS sessions on the CAS server, and these CAS sessions are independent of each other. One reason you would want to create multiple CAS sessions is to execute programs in parallel. One example would be if you want to execute three different machine learning models at the same time. Otherwise, if you do not terminate your connection to CAS, you might mistakenly create multiple connections to CAS, consuming resources.

To connect to the CAS server, you need:

- the host name,
- the port number,
- authentication

Be aware that connecting to the CAS server can be implemented in various ways, so you might need to see your system administrator about how to make a connection. Please follow company policy regarding authentication.

In [16]:
```
## my personal environment connection information to SAS Viya (will not work in your e
try:
    path = os.getenv('CAS_CREDENTIALS')
    pem_file = os.getenv('CAS_CLIENT_SSL_CA_LIST')
    conn = CASAuth(path, ssl_ca_list = pem_file)
except:
    print('Use your own connection information with the swat.CAS method')


##############################################################################
## General syntax
##############################################################################
## conn = swat.CAS('server.demo.sas.com', port number, 'username', 'password')
##############################################################################


##############################################################################
## Viya for Learners 3.5 CAS connection information
##############################################################################
# hostValue = os.environ.get('CASHOST')
# portValue = os.environ.get('CASPORT')
# passwordToken=os.environ.get('SAS_VIYA_TOKEN')
# conn = swat.CAS(hostname=hostValue,port=portValue,password=passwordToken)
```

```
The access token may have expired - attempting to refresh the token
The access token was stored for you in the access_token.txt file. The access token ex
pires in 24 hours.
CAS Connection created
```

REQUIREMENT: Add your connection information below

In [17]:
```
## conn = swat.CAS('server.demo.sas.com', port number, 'username', 'password')
```

View the type of the **conn** object.

In [18]:
```
type(conn)
```

Out[18]:
```
casauth.casauth.CASAuth
```

Confirm the connection is working and view the version of SAS Viya.

In [19]:
```
try:
    print(conn.about()['About']['Viya Version'])  ## SAS Viya version
```

```
except:
    print(conn.about()['About']['Version'])       ## SAS Viya 3.5 VFL version
```

```
NOTE: Grid node action status report: 5 nodes, 9 total actions executed.
Stable 2023.08
```

c. Set up demonstration data

Create the demonstration data. The demonstration data comes from the **Samples** caslib in SAS Viya. The **Samples** caslib is included in SAS Viya installations. The data is a small sample table for training purposes.

In [20]:
```
def setup_demo_data(datasourcefile, incaslib, outfilename, outcaslib):
    '''
    Function
    - loads a server-side file into memory (datasourcefile) from a specific caslib (ir
    - saves the demo_data CAS table from the Casuser caslib as the specific data sourc
    - drops the demo_data CAS table from the Casuser caslib after it has been saved as
    '''

    # Load data into memory abd name it demo_data in the casuser caslib
    conn.loadTable(path = datasourcefile, caslib = incaslib,
                   casout = {'name':'demo_data', 'caslib':'casuser', 'replace':True})

    # Save the demo_data CAS table back to disk
    conn.save(table = {'name':'demo_data', 'caslib':'casuser'},
              name = outfilename, caslib = outcaslib, replace=True)

    # Drop the demo_data CAS table
    conn.dropTable('demo_data', caslib = 'casuser')

##
## Load demo data
##
setup_demo_data(datasourcefile='WARRANTY_CLAIMS_0117.sashdat', incaslib='samples',
                outfilename='warranty_demo.csv', outcaslib='casuser')
```

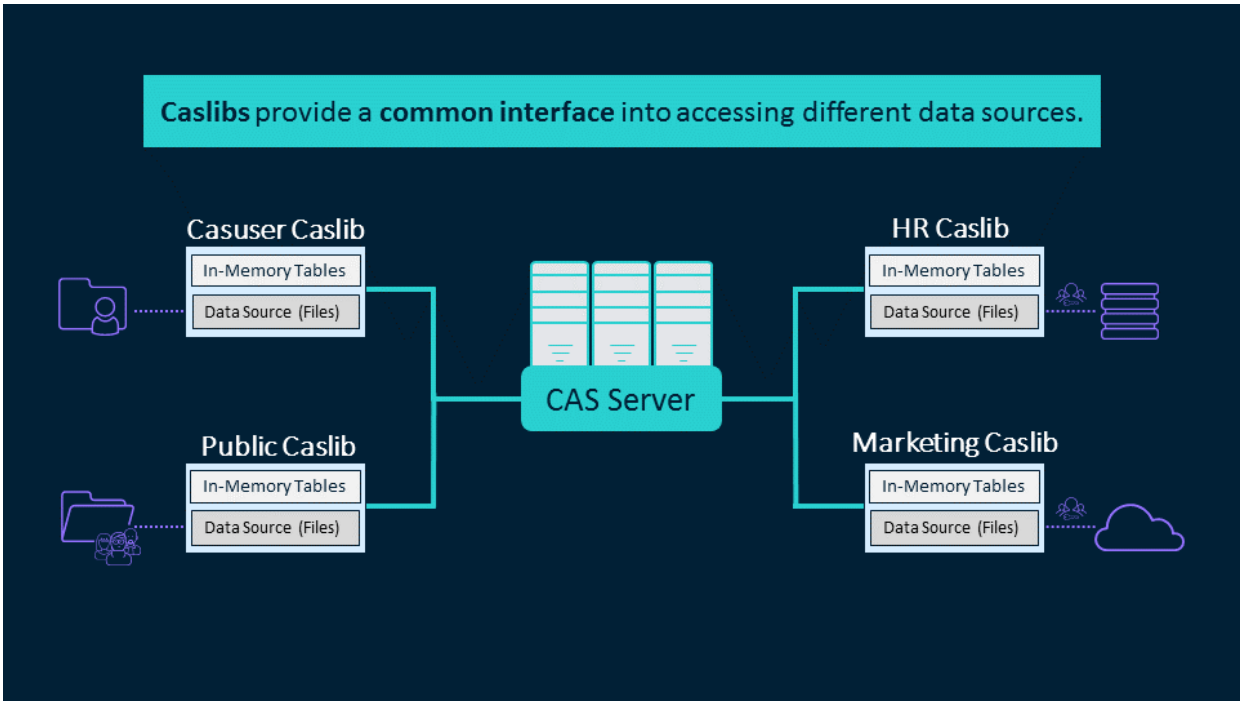
```
NOTE: Cloud Analytic Services made the file WARRANTY_CLAIMS_0117.sashdat available as
table DEMO_DATA in caslib CASUSER(Peter.Styliadis@sas.com).
NOTE: Cloud Analytic Services saved the file warranty_demo.csv in caslib CASUSER(Pete
r.Styliadis@sas.com).
NOTE: Cloud Analytic Services dropped table demo_data from caslib CASUSER(Peter.Styli
adis@sas.com).
```

2. Explore available data in the CAS server

SAS Viya Data Connectors contain connection information and data-source specifics to connect with data sources throughout your organization.

In [21]:
```
display_image('11a_caslibs.png')
```

Out[21]:



CAS stores and accesses data via caslibs. Caslibs provide a common interface into accessing different data sources. No matter which data source you access in CAS, it is connected via a caslib. Caslibs connect into a variety of data sources in your organization.

A caslib consists of three main parts.

First, there's the data source portion. The data source portion contains connection information to a physical data source for storage. Data might be stored on disk in a path with familiar formats such as SAS data sets, CSV, Microsoft Excel, or parquet files, or stored in databases, the cloud, Hadoop, or other systems. The data source portion stores the connection and authorization information required for interacting with those files using SAS Viya data connectors. Data stored in the data source portion of a caslib is generally referred to as files. Files in the data source area cannot be processed directly.

The second part of a caslib is the in-memory portion. The in-memory portion provides a space to hold tables that can have operations performed on them. You can almost think of a CAS table as a DataFrame on the CAS server. A file must first be loaded into memory from a data source. Data loaded into memory is generally referred to as tables or CAS tables. >When they are loaded, in-memory tables are available for processing, and they persist in memory. Because there's no connection between the in-memory copy and the data source file, if you want to permanently keep changes made to the data during processing, the in-memory table should be explicitly written back to the data source. If you are done with the CAS table, you can drop the table. We discuss this more later.

Lastly, a caslib contains access controls about who can access that specific caslib.

a. View available caslibs (data sources) in the CAS server

In [22]:
```
conn.caslibInfo()
```

Out[22]: **§ CASLibInfo**

	Name	Type	Description	
0	AIoTPgMeta	PATH	Stores quality analytic suite postgres tables.	/cas/data/caslibs/aiot
1	CASUSER(Peter.Styliadis@sas.com)	PATH	Personal File System Caslib	/cas/data/caslibs/casuserlibraries/peter.styliadis@
2	cpgretl	PATH		/cas/data/caslibs/c
3	CPSAppData	PATH		/cas/data/caslibs/CPSA
4	EDUPub	PATH		/cas/data/caslibs/
5	EP_CommunityCollege	PATH		/cas/data/caslibs/educationpractice/EP_Community
6	EP_DOE	PATH		/cas/data/caslibs/educationpractice/
7	EP_Forecasting	PATH		/cas/data/caslibs/educationpractice/EP_For
8	EP_K12	PATH		/cas/data/caslibs/educationpractice
9	EP_SAfE	PATH		/cas/data/caslibs/educationpractice/
10	EP_Shapes	PATH		/cas/data/caslibs/educationpractice/EP
11	EP_Sports	PATH		/cas/data/caslibs/educationpractice/EP
12	EP_University	PATH		/cas/data/caslibs/educationpractice/EP_U
13	EP_Workforce	PATH		/cas/data/caslibs/educationpractice/EP_W
14	Formats	PATH	Stores user defined formats.	/cas/data/caslibs/
15	GPCI	PATH	Group Lib for Global CI team. Suneel Grover is the main contact	/cas/data/caslibs/
16	GTPPub	PATH		/cas/data/caslibs/
17	images	PATH		/cas/data/caslibs
18	IoTPub	PATH		/cas/data/caslibs
19	jsmdata	PATH	RITM0602947 - Fang K. Chen: We would like to leverage SSEMonthly in support of the "2023 Joint Statistical Meetings Conference".	/cas/data/caslibs/

	Name	Type	Description	
20	ModelPerformanceData	PATH	Stores performance data output for the Model Management service.	/cas/data/caslibs/modelMonitc
21	Models	PATH	Stores models created by Visual Analytics for use in other analytics or SAS Studio.	/cas/data/caslibs/
22	PMVAPub	PATH		/cas/data/caslibs/PI
23	Public	PATH	Shared and writeable caslib, accessible to all users. 30 days of data. Purged Monthly.	/cas/data/caslib
24	QASANLOUT	PATH	The CASLib would be used to store the analysis output datasets.	/cas/data/caslibs/qa
25	QASMartStore	PATH	Stores quality analytic suite mart tables.	/cas/data/caslibs/qasM
26	Samples	PATH	Stores sample data, supplied by SAS.	/cas/data/caslibs/
27	Samples2023	PATH		/cas/data/caslibs/Samp
28	sashack22	PATH		/cas/data/caslibs/sa
29	sashack23	PATH		/cas/data/caslibs/sa
30	SGF2021	PATH	Private CASLib for SGF 2021 Content Admins	/cas/data/caslibs/
31	SGF21AAI	PATH		/cas/data/caslibs/
32	SIS2021_6	PATH		/cas/data/caslibs/sis_2021/SI
33	swee	PATH		/cas/data/casli
34	UKI	PATH		/cas/data/cas

elapsed 0.0086s · user 0.00734s · sys 0.00605s · mem 1.84MB

b. View available data source files in a caslib

In [23]:
```
conn.fileInfo(caslib = 'samples')
```

Out[23]: **§ FileInfo**

	Permission	Owner	Group	Name	Size	Encryption	
0	-rwxr-xr-x	sas	sas	COSTCHANGE.sashdat	9776	NONE	18T1
1	-rwxr-xr-x	sas	sas	PROMO_EFFECTIVENESS_X_EFFECTS_2.sashdat	9312	NONE	18T1
2	-rwxr-xr-x	sas	sas	RAND_RETAILDEMO.sashdat	42612664	NONE	18T1
3	-rwxr-xr-x	sas	sas	SW_LAKE_RT_SENSOR_WATERQUALITY.sashdat	301312	NONE	18T1
4	-rwxr-xr-x	sas	sas	WARRANTY_CLAIMS_0117.sashdat	13563272	NONE	18T1
5	-rwxr-xr-x	sas	sas	WATER_CLUSTER.sashdat	773208	NONE	18T1
6	-rwxr-xr-x	sas	sas	predef_svrtdist.sashdat	78872	NONE	08T0

elapsed 0.0322s · user 0.00412s · sys 0.0107s · mem 1.81MB

In [24]:
```
conn.fileInfo(caslib = 'casuser')
```

Out[24]: **§ FileInfo**

	Permission	Owner	Group	Name	Size	Encryption	Time
0	-rwxr-xr-x	sas	sas	cars.sas7bdat	139264		2023-02-23T14:21:31+00:00
1	-rwxr-xr-x	sas	sas	previoussales.sas7bdat	73728		2023-04-26T20:22:48+00:00
2	-rwxr-xr-x	sas	sas	VTI.sashdat	413080	NONE	2022-10-11T13:40:38+00:00
3	-rwxr-xr-x	sas	sas	hmeq.sashdat	630384	NONE	2022-10-13T17:56:59+00:00
4	-rwxr-xr-x	sas	sas	tsa_claims_raw.csv	34936205		2023-09-18T13:09:17+00:00
5	-rwxr-xr-x	sas	sas	warranty_demo.csv	53297896		2023-09-19T13:40:52+00:00
6	-rwxr-xr-x	sas	sas	warranty_final.sashdat	116882608	NONE	2023-09-18T15:25:48+00:00
7	-rwxr-xr-x	sas	sas	cars.parquet	4096	NONE	2022-11-17T14:19:19+00:00
8	-rwxr-xr-x	sas	sas	RAND_RETAILDEMO.csv	240072190		2023-05-30T13:10:38+00:00
9	-rwxr-xr-x	sas	sas	warranty_final.csv	43615117		2023-05-30T13:52:57+00:00
10	-rwxr-xr-x	sas	sas	cars.sashdat	102936	NONE	2023-05-09T18:03:20+00:00
11	-rwxr-xr-x	sas	sas	cars.csv	34289		2023-05-09T18:03:20+00:00
12	-rwxr-xr-x	sas	sas	cars.txt	34289		2023-05-09T18:03:20+00:00
13	-rwxr-xr-x	sas	sas	home_equity_raw.parquet	4096	NONE	2023-05-24T17:41:10+00:00
14	-rwxr-xr-x	sas	sas	previousSales.sashdat	8416	NONE	2023-04-26T20:22:07+00:00
15	-rwxr-xr-x	sas	sas	products.csv	492125		2023-01-18T12:49:50+00:00
16	-rwxr-xr-x	sas	sas	discount_dim.sashdat	237952	NONE	2023-01-18T12:49:51+00:00
17	-rwxr-xr-x	sas	sas	heart.csv	458623		2023-01-20T20:41:13+00:00
18	-rwxr-xr-x	sas	sas	heart_raw.csv	458623		2023-01-20T20:43:00+00:00
19	-rwxr-xr-x	sas	sas	finalRetail.sashdat	410194024	NONE	2023-05-30T13:53:44+00:00
20	-rwxr-xr-x	sas	sas	MULTIPLICATION.sashdat	17832	NONE	2023-06-06T14:21:06+00:00

	Permission	Owner	Group	Name	Size	Encryption	Time
21	-rwxr-xr-x	sas	sas	warranty_final.parquet	4096	NONE	2023-05-09T18:16:04+00:00
22	-rwxr-xr-x	sas	sas	gov_it_budget_wide.sashdat	7736512	NONE	2023-08-16T14:52:11+00:00
23	-rwxr-xr-x	sas	sas	warranty_claims.parquet	4096	NONE	2023-06-02T15:11:24+00:00
24	-rwxr-xr-x	sas	sas	warranty_claims_test.parquet	4096	NONE	2023-06-02T15:17:55+00:00
25	-rwxr-xr-x	sas	sas	warranty_demo.parquet	4096	NONE	2023-06-05T18:39:16+00:00
26	-rwxr-xr-x	sas	sas	loans_raw.sashdat	13821560	NONE	2023-02-23T14:21:28+00:00
27	-rwxr-xr-x	sas	sas	customers_raw.csv	3101649		2023-02-23T14:21:30+00:00
28	-rwxr-xr-x	sas	sas	appRatings.sashdat	3954240	NONE	2023-02-23T14:21:30+00:00
29	-rwxr-xr-x	sas	sas	heart.sashdat	1020320	NONE	2023-02-23T14:21:31+00:00
30	-rwxr-xr-x	sas	sas	2023-05-29_wordle_wide.csv	463936		2023-05-29T11:14:20+00:00
31	-rwxr-xr-x	sas	sas	2023-05-29_wordle_long.csv	1515237		2023-05-29T11:14:21+00:00
32	-rwxr-xr-x	sas	sas	RAND_RETAILDEMO.parquet	4096	NONE	2023-06-05T18:39:37+00:00
33	-rwxr-xr-x	sas	sas	MY_FUNCTIONS.sashdat	30000	NONE	2023-09-18T13:45:18+00:00
34	-rwxr-xr-x	sas	sas	sales_final.parquet	8192	NONE	2023-06-16T16:08:08+00:00
35	-rwxr-xr-x	sas	sas	gov_it_budget_narrow.sashdat	37866240	NONE	2023-08-16T14:52:12+00:00
36	-rwxr-xr-x	sas	sas	final_sales.parquet	4096	NONE	2023-08-22T18:31:09+00:00
37	-rwxr-xr-x	sas	sas	final_sales_2023-08-23.parquet	4096	NONE	2023-08-23T18:02:25+00:00
38	-rwxr-xr-x	sas	sas	final_sales_2023-08-25.parquet	4096	NONE	2023-08-25T19:38:58+00:00
39	-rwxr-xr-x	sas	sas	FUNCTIONS.sashdat	19184	NONE	2023-09-09T00:05:38+00:00
40	-rwxr-xr-x	sas	sas	MY_UDFS.sashdat	35408	NONE	2023-09-09T00:23:12+00:00
41	-rwxr-xr-x	sas	sas	TEST_UDFS.sashdat	28648	NONE	2023-09-08T21:41:38+00:00

elapsed 0.0804s . user 0.0125s . sys 0.00607s . mem 1.82MB

c. View available in-memory CAS tables in a caslib

In [25]:
```
conn.tableInfo(caslib = 'samples')
```

Out[25]: **§ TableInfo**

	Name	Label	Rows	Columns	IndexedColumn
0	RAND_RETAILDEMO		930046	40	
1	PROMO_EFFECTIVENESS_X_EFFECTS_2	promo_effectiveness_x_effects_2	14	11	
2	COSTCHANGE		32	10	
3	WARRANTY_CLAIMS_0117		153217	42	

elapsed 0.0094s · user 0.0122s · sys 0.00314s · mem 1.9MB

In [17]:
```
conn.tableInfo(caslib = 'casuser')
```

Out[17]: **§ TableInfo**

	Name	Rows	Columns	IndexedColumns	Encoding	CreateTimeFormatted	Mod
0	GOV_IT_BUDGET_NARROW	13555	6	0	utf-8	2023-08-01T17:19:34+00:00	01

elapsed 0.00919s · user 0.00732s · sys 0.011s · mem 1.91MB

3. Load a data source file into memory

a. Load a server-side file into memory

Confirm the **warranty_demo.csv** file is available.

In [26]:
```
conn.fileInfo(caslib = 'casuser', path='warranty_demo.csv')
```

Out[26]: **§ FileInfo**

	Permission	Owner	Group	Name	Size	Encryption	Time	ModTir
0	-rwxr-xr-x	sas	sas	warranty_demo.csv	53297896		2023-09-19T13:40:52+00:00	2.010750e+

elapsed 0.017s · user 0.00894s · sys 0.00621s · mem 1.8MB

Load the **warranty_demo.csv** data source file into memory into the **Casuser** caslib. This is similar to loading a file as a DataFrame in pandas.

In [27]:
```
conn.loadTable(path = 'warranty_demo.csv', caslib = 'casuser',    ## Specify the data s
               importOptions = {'fileType':'CSV',                 ## Import file specif
                                'guessRows':10000},
               casOut = {'name':'warranty_demo',                  ## Specify the output
                         'caslib':'casuser',
                         'replace':True})
```

```
NOTE: Cloud Analytic Services made the file warranty_demo.csv available as table WARR
ANTY_DEMO in caslib CASUSER(Peter.Styliadis@sas.com).
```

Out[27]:

§ caslib

CASUSER(Peter.Styliadis@sas.com)

---

§ tableName

WARRANTY_DEMO

---

§ casTable

CASTable('WARRANTY_DEMO', caslib='CASUSER(Peter.Styliadis@sas.com)')

elapsed 0.472s · user 1.41s · sys 0.565s · mem 505MB

View the available in-memory tables on the CAS server. The **WARRANTY_DEMO** CAS table is in-memory on the CAS server.

In [28]:
```
conn.tableInfo(caslib = 'casuser')
```

Out[28]:

§ TableInfo

	Name	Rows	Columns	IndexedColumns	Encoding	CreateTimeFormatted	ModTimeF
0	WARRANTY_DEMO	153217	42	0	utf-8	2023-09-19T13:42:06+00:00	19T13:42:

elapsed 0.00834s · user 0.00204s · sys 0.01s · mem 1.84MB

b. Explore the CAS table

Reference the CAS table.

In [29]:
```
castbl = conn.CASTable('WARRANTY_DEMO', caslib = 'casuser')
castbl
```

Out[29]:
```
CASTable('WARRANTY_DEMO', caslib='casuser')
```

View the object type.

In [30]:
```
type(castbl)
```

Out[30]:
```
swat.cas.table.CASTable
```

View the attributes of the CAS table.

In [31]:
```
castbl.shape
```

Out[31]:
```
(153217, 42)
```

View details about the CAS table using the tableDetails CAS action.

In [32]:
```
castbl.tableDetails()
```

Out[32]:

§ TableDetails

	Node	Blocks	Active	Rows	IndexSize	DataSize	VardataSize	CompressedSize	CompressionRa
0	ALL	210	105	153217	0	104045473	28049841	0	

elapsed 0.0268s · user 0.00519s · sys 0.0236s · mem 1.81MB

View the column attributes of a CAS table.

In [33]:
```
castbl.columnInfo()
```

Out[33]: **§ ColumnInfo**

	Column	Label	ID	Type	RawLength	FormattedLength	Format	NFL	NI
0	claim_attribute_1		1	varchar	6	6		0	
1	seller_attribute_5		2	double	8	12		0	
2	product_attribute_1		3	varchar	2	2		0	
3	product_attribute_2		4	varchar	2	2		0	
4	product_attribute_3		5	double	8	12		0	
5	product_attribute_4		6	varchar	6	6		0	
6	product_attribute_5		7	double	8	12		0	
7	customer_attribute_1		8	varchar	6	6		0	
8	x_product_attribute7		9	varchar	10	10		0	
9	x_product_attribute9		10	varchar	10	10		0	
10	srvc_prvdr_attribute_1		11	varchar	24	24		0	
11	srvc_prvdr_attribute_4		12	varchar	16	16		0	
12	srvc_prvdr_attribute_2		13	double	8	12		0	
13	srvc_prvdr_attribute_3		14	varchar	6	6		0	
14	srvc_prvdr_attribute_5		15	varchar	4	4		0	
15	primary_labor_group_desc		16	varchar	19	19		0	
16	primary_labor_desc		17	varchar	50	50		0	
17	total_claim_count		18	double	8	12		0	
18	CLAIM_REPAIR_START_DT_SK		19	double	8	12		0	
19	CLAIM_REPAIR_END_DT_SK		20	double	8	12		0	
20	CLAIM_PROCESSED_DT_SK		21	double	8	12		0	
21	CLAIM_SUBMITTED_DT_SK		22	double	8	12		0	
22	PRIMARY_LABOR_CD		23	varchar	5	5		0	
23	DEFECT_SK		24	double	8	12		0	
24	PRIMARY_LABOR_GROUP_CD		25	varchar	1	1		0	
25	PRIMARY_MATERIAL_ID		26	varchar	5	5		0	
26	PRIMARY_MATERIAL_GROUP_CD		27	double	8	12		0	
27	WARRANTY_CLAIM_ID		28	varchar	29	29		0	
28	USAGE_VALUE		29	double	8	12		0	
29	GROSS_CLAIM_AMT		30	double	8	12		0	
30	GROSS_LABOR_AMT		31	double	8	12		0	
31	GROSS_MATERIAL_AMT		32	double	8	12		0	
32	GROSS_OTHER_AMT		33	double	8	12		0	

	Column	Label	ID	Type	RawLength	FormattedLength	Format	NFL	NF
33	PRODUCT_ID		34	varchar	17	17		0	
34	PRODUCT_MODEL_CD		35	varchar	8	8		0	
35	PRODUCTION_DT		36	double	8	12		0	
36	IN_SERVICE_DT		37	double	8	12		0	
37	SHIP_DT		38	double	8	12		0	
38	SHIP_YEAR_CD		39	double	8	12		0	
39	DEFECT_CD		40	varchar	0	1		0	
40	service_provider_latitude		41	double	8	12		0	
41	service_provider_longitude		42	double	8	12		0	

elapsed 0.0109s · user 0.00863s · sys 0.00896s · mem 2.49MB

Preview the distributed CAS table.

In [34]:
```
castbl.head()
```

Out[34]:

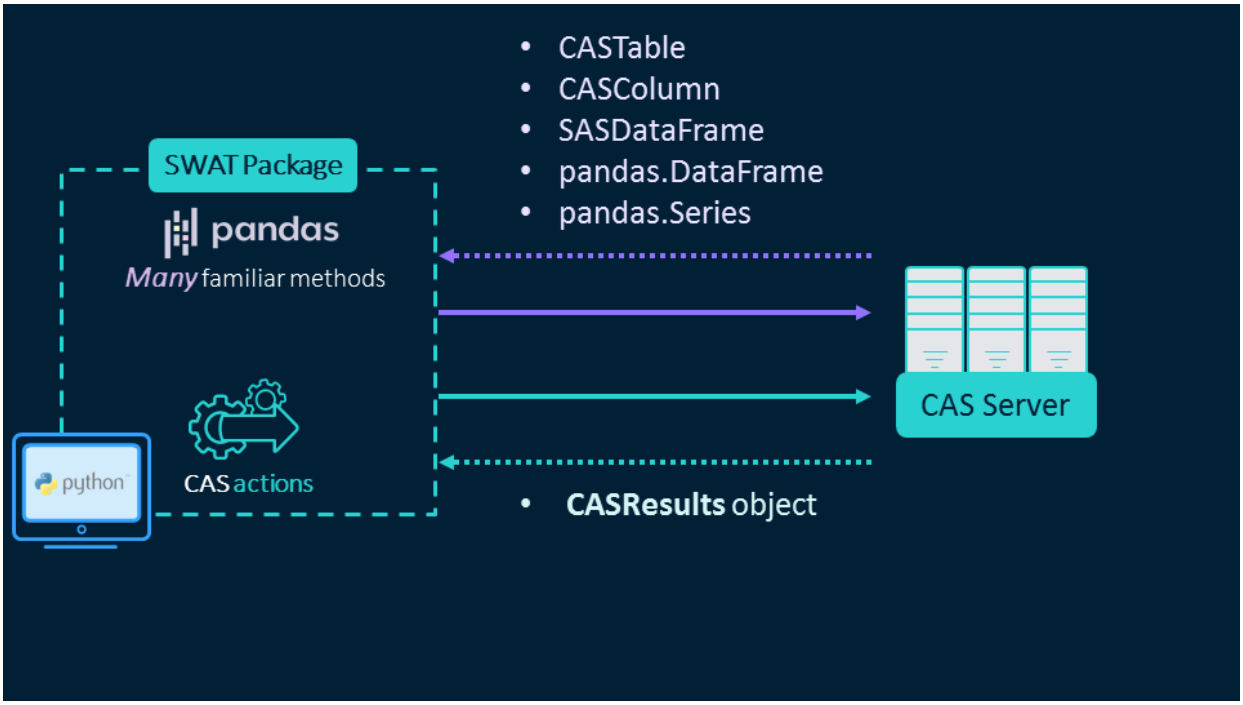
	claim_attribute_1	seller_attribute_5	product_attribute_1	product_attribute_2	product_attribute_3	p
0	Type 4	91.0	F1	XE	110.0	
1	Type 4	283.0	F1	XE	104.0	
2	Type 4	49.0	I1	XE	110.0	
3	Type 4	49.0	J1	XE	110.0	
4	Type 4	49.0	I1	XE	110.0	

c. SWAT package objects overview

CASTable vs. DataFrame vs. SASDataFrame

In [35]:
```
display_image('12a_swat_objects.png')
```

Out[35]:



Because the SWAT package tries to blend the world of Pandas and CAS into a single world, you have to be aware of whether you are calling a CAS action or a Pandas API method from the SWAT package because the objects returned to the client will differ.

When you execute Pandas methods from the SWAT package, CAS can return a variety of objects to the client, such as a CASTable, CASColumn, SASDataFrame, pandas.DataFrame, or pandas.Series.

When you execute a CAS action, CAS always returns a CASResults object (Python dictionary) to the Python client.

CASTable object

CASTable objects and DataFrame objects (either pandas.DataFrame or SASDataFrame) act very similar in many ways, but they are extremely different constructs. CASTable objects do not contain actual data. They are simply a client-side view of the data in a CAS table on a CAS server. DataFrames and SASDataFrames contain data in-memory on the client machine.

In [36]:
```
type(castbl)
```

Out[36]:
```
swat.cas.table.CASTable
```

In [37]:
```
castbl
```

Out[37]:
```
CASTable('WARRANTY_DEMO', caslib='casuser')
```

SASDataFrame object

The SASDataFrame object is a simple subclass of pandas.DataFrame and local to the client. It merely adds attributes to hold SAS metadata such as titles, labels, column metadata, etc. It also

adds a few utility methods for handling By group representations.

In [38]:
```
df = castbl.head()

display(type(df), df)
```

```
swat.dataframe.SASDataFrame
```

	claim_attribute_1	seller_attribute_5	product_attribute_1	product_attribute_2	product_attribute_3	p
0	Type 4	91.0	F1	XE	110.0	
1	Type 4	283.0	F1	XE	104.0	
2	Type 4	49.0	I1	XE	110.0	
3	Type 4	49.0	J1	XE	110.0	
4	Type 4	49.0	I1	XE	110.0	

CASResults object

The output of all CAS actions is a CASResults object. This is a Python ordered dictionary with a few methods added to assist in handling the output keys, and attributes added to report information from the CAS action.

In [39]:
```
cr = castbl.fetch(to = 5)

display(type(cr), cr)
```

```
swat.cas.results.CASResults
```

§ Fetch

	claim_attribute_1	seller_attribute_5	product_attribute_1	product_attribute_2	product_attribute_3	p
0	Type 4	91.0	F1	XE	110.0	
1	Type 4	283.0	F1	XE	104.0	
2	Type 4	49.0	I1	XE	110.0	
3	Type 4	49.0	J1	XE	110.0	
4	Type 4	49.0	I1	XE	110.0	

elapsed 0.016s · user 0.00341s · sys 0.0276s · mem 5.24MB

View the keys in the CASResults object.

In [40]:
```
cr.keys()
```

Out[40]:
```
odict_keys(['Fetch'])
```

Access the SASDataFrame from the *Fetch* key.

In [41]:
```
df = cr['Fetch']

display(type(df), df)
```

```
swat.dataframe.SASDataFrame
```

	claim_attribute_1	seller_attribute_5	product_attribute_1	product_attribute_2	product_attribute_3	p
0	Type 4	91.0	F1	XE	110.0	
1	Type 4	283.0	F1	XE	104.0	
2	Type 4	49.0	I1	XE	110.0	
3	Type 4	49.0	J1	XE	110.0	
4	Type 4	49.0	I1	XE	110.0	

d. Rename columns, drop columns, format date columns

Get the name of all the CAS table columns.

In [42]:
```
castableColumnNames = castbl.columns.to_list()
castableColumnNames
```

Out[42]:
```
['claim_attribute_1',
 'seller_attribute_5',
 'product_attribute_1',
 'product_attribute_2',
 'product_attribute_3',
 'product_attribute_4',
 'product_attribute_5',
 'customer_attribute_1',
 'x_product_attribute7',
 'x_product_attribute9',
 'srvc_prvdr_attribute_1',
 'srvc_prvdr_attribute_4',
 'srvc_prvdr_attribute_2',
 'srvc_prvdr_attribute_3',
 'srvc_prvdr_attribute_5',
 'primary_labor_group_desc',
 'primary_labor_desc',
 'total_claim_count',
 'CLAIM_REPAIR_START_DT_SK',
 'CLAIM_REPAIR_END_DT_SK',
 'CLAIM_PROCESSED_DT_SK',
 'CLAIM_SUBMITTED_DT_SK',
 'PRIMARY_LABOR_CD',
 'DEFECT_SK',
 'PRIMARY_LABOR_GROUP_CD',
 'PRIMARY_MATERIAL_ID',
 'PRIMARY_MATERIAL_GROUP_CD',
 'WARRANTY_CLAIM_ID',
 'USAGE_VALUE',
 'GROSS_CLAIM_AMT',
 'GROSS_LABOR_AMT',
 'GROSS_MATERIAL_AMT',
 'GROSS_OTHER_AMT',
 'PRODUCT_ID',
 'PRODUCT_MODEL_CD',
 'PRODUCTION_DT',
 'IN_SERVICE_DT',
 'SHIP_DT',
 'SHIP_YEAR_CD',
 'DEFECT_CD',
 'service_provider_latitude',
 'service_provider_longitude']
```

Specify the new column names as a list.

In [43]:
```
newColumnNames = [
            'campaign_type','selling_dealer','vehicle_class','platform','trim_level','
            'vehicle_assembly_plant','repairing_division','repairing_state_province','
            'primary_labor_group','primary_labor_description','total_claim_count','cla
            'claim_submitted_date','primary_labor_code','defect_key','primary_labor_gr
            'primary_replaced_material_group_code','warranty_claim_id','usage_value','
            'gross_other_amount','product_unit_id','product_model','product_unit_assem
            'latitude','longitude'
]
```

Create a list of dictionaries to rename the columns in the CAS table.

In [44]:
```
updateColumns = []

## Create a list of dictionaries. Each dictionary specifies the current column name, d
for position in range(len(castableColumnNames)):

    # Rename columns
    colUpdate = {'name' : castableColumnNames[position], 'rename' : newColumnNames[pos

    # # Add date format to date columns
    if colUpdate['rename'].find('date') > 0:
        colUpdate['format'] = 'DATE9.'

    # Append dictionaries to list
    updateColumns.append(colUpdate)

updateColumns
```

Out[44]:
```
[{'name': 'claim_attribute_1', 'rename': 'campaign_type'},
 {'name': 'seller_attribute_5', 'rename': 'selling_dealer'},
 {'name': 'product_attribute_1', 'rename': 'vehicle_class'},
 {'name': 'product_attribute_2', 'rename': 'platform'},
 {'name': 'product_attribute_3', 'rename': 'trim_level'},
 {'name': 'product_attribute_4', 'rename': 'make'},
 {'name': 'product_attribute_5', 'rename': 'model_year'},
 {'name': 'customer_attribute_1', 'rename': 'customer_country'},
 {'name': 'x_product_attribute7', 'rename': 'engine_model'},
 {'name': 'x_product_attribute9', 'rename': 'vehicle_assembly_plant'},
 {'name': 'srvc_prvdr_attribute_1', 'rename': 'repairing_division'},
 {'name': 'srvc_prvdr_attribute_4', 'rename': 'repairing_state_province'},
 {'name': 'srvc_prvdr_attribute_2', 'rename': 'repairing_region'},
 {'name': 'srvc_prvdr_attribute_3', 'rename': 'repairing_country'},
 {'name': 'srvc_prvdr_attribute_5', 'rename': 'repairing_dealer'},
 {'name': 'primary_labor_group_desc', 'rename': 'primary_labor_group'},
 {'name': 'primary_labor_desc', 'rename': 'primary_labor_description'},
 {'name': 'total_claim_count', 'rename': 'total_claim_count'},
 {'name': 'CLAIM_REPAIR_START_DT_SK',
  'rename': 'claim_repair_start_date',
  'format': 'DATE9.'},
 {'name': 'CLAIM_REPAIR_END_DT_SK',
  'rename': 'claim_repair_end_date',
  'format': 'DATE9.'},
 {'name': 'CLAIM_PROCESSED_DT_SK',
  'rename': 'claim_processed_date',
  'format': 'DATE9.'},
 {'name': 'CLAIM_SUBMITTED_DT_SK',
  'rename': 'claim_submitted_date',
  'format': 'DATE9.'},
 {'name': 'PRIMARY_LABOR_CD', 'rename': 'primary_labor_code'},
 {'name': 'DEFECT_SK', 'rename': 'defect_key'},
 {'name': 'PRIMARY_LABOR_GROUP_CD', 'rename': 'primary_labor_group_code'},
 {'name': 'PRIMARY_MATERIAL_ID', 'rename': 'primary_replaced__material_id'},
 {'name': 'PRIMARY_MATERIAL_GROUP_CD',
  'rename': 'primary_replaced_material_group_code'},
 {'name': 'WARRANTY_CLAIM_ID', 'rename': 'warranty_claim_id'},
 {'name': 'USAGE_VALUE', 'rename': 'usage_value'},
 {'name': 'GROSS_CLAIM_AMT', 'rename': 'gross_claim_amount'},
 {'name': 'GROSS_LABOR_AMT', 'rename': 'gross_labor_amount'},
 {'name': 'GROSS_MATERIAL_AMT', 'rename': 'gross_material_amount'},
 {'name': 'GROSS_OTHER_AMT', 'rename': 'gross_other_amount'},
 {'name': 'PRODUCT_ID', 'rename': 'product_unit_id'},
 {'name': 'PRODUCT_MODEL_CD', 'rename': 'product_model'},
 {'name': 'PRODUCTION_DT',
  'rename': 'product_unit_assembly_date',
  'format': 'DATE9.'},
 {'name': 'IN_SERVICE_DT', 'rename': 'service_year_date', 'format': 'DATE9.'},
 {'name': 'SHIP_DT', 'rename': 'ship_date', 'format': 'DATE9.'},
 {'name': 'SHIP_YEAR_CD', 'rename': 'ship_year'},
 {'name': 'DEFECT_CD', 'rename': 'defect_code'},
 {'name': 'service_provider_latitude', 'rename': 'latitude'},
 {'name': 'service_provider_longitude', 'rename': 'longitude'}]
```

Specify a list of columns to drop.

In [45]:
```
dropColumns = ['defect_code','repairing_division','usage_value','campaign_type','custo
               'product_unit_assembly_date','primary_replaced_material_group_code','pr
               'selling_dealer','vehicle_class','ship_date', 'total_claim_count']
```

Use the alterTable CAS action to rename, drop and rearrange columns.

In [46]:
```
castbl.alterTable(columns = updateColumns,
                  drop = dropColumns)
```

Out[46]: elapsed 0.00937s · user 0.0134s · sys 0.00571s · mem 2.37MB

Return a tuple representing the dimensionality of the CASTable.

In [47]:
```
castbl.shape
```

Out[47]: (153217, 29)

Preview the 5 rows of the CAS table.

In [48]:
```
castbl.head()
```

Out[48]:

	platform	trim_level	make	model_year	engine_model	vehicle_assembly_plant	repairing_state_prov
0	XE	110.0	Zeus	2018.0	8 cylinder	Charlotte	Ge
1	XE	104.0	Zeus	2018.0	4 cylinder	Detroit	Ge
2	XE	110.0	Zeus	2016.0	4 cylinder	Pittsburgh	Ge
3	XE	110.0	Zeus	2017.0	8 cylinder	Detroit	Ge
4	XE	110.0	Zeus	2016.0	4 cylinder	Pittsburgh	Ge

Rearrange the CAS table column order.

In [49]:
```
## Specify the new column order
newColumnOrder = ['warranty_claim_id', 'make', 'product_model', 'model_year', 'platfor
                  'primary_labor_group', 'primary_labor_description', 'primary_labor_c
                  'gross_labor_amount', 'gross_material_amount', 'gross_other_amount',
                  'repairing_country', 'repairing_dealer', 'latitude', 'longitude', 'c
                  'claim_submitted_date','service_year_date']

## Rearrange the columns in the CAS table
castbl.alterTable(columnOrder = newColumnOrder)

## Preview the new CAS table
castbl.head()
```

Out[49]:

	warranty_claim_id	make	product_model	model_year	platform	trim_level	engine_
0	C1_85809801_4V4N19TG75N374989	Zeus	Galacto	2018.0	XE	110.0	8 c
1	C1_85809801_4V4N19TG75N374989	Zeus	Galacto	2018.0	XE	104.0	4 c
2	C1_12730601_4V4NC9TH53N336841	Zeus	Galacto	2016.0	XE	110.0	4 c
3	C1_10018401_4V4NC9GH54N351397	Zeus	Galacto	2017.0	XE	110.0	8 c
4	C1_15415501_4V4NC9TH33N337387	Zeus	Galacto	2016.0	XE	110.0	4 c

4. Exploratory data analysis

a. Distinct and missing values

View the unique values of a column.

In [50]:
```
castbl.make.unique()
```

Out[50]:
```
array(['Apollo', 'Ares', 'Titan', 'Zeus'], dtype=object)
```

Use the distinct CAS action to compute the distinct and missing number of values of the columns.

In [51]:
```
distinct_df = castbl.distinct()['Distinct']

display(type(distinct_df),distinct_df)
```

```
swat.dataframe.SASDataFrame
```

Distinct Counts for WARRANTY_DEMO

	Column	NDistinct	NMiss	Trunc
0	warranty_claim_id	144681.0	0.0	0.0
1	make	4.0	0.0	0.0
2	product_model	21.0	0.0	0.0
3	model_year	5.0	0.0	0.0
4	platform	2.0	0.0	0.0
5	trim_level	60.0	0.0	0.0
6	engine_model	4.0	0.0	0.0
7	vehicle_assembly_plant	3.0	0.0	0.0
8	primary_labor_group	8.0	0.0	0.0
9	primary_labor_description	80.0	0.0	0.0
10	primary_labor_code	80.0	0.0	0.0
11	defect_key	745.0	0.0	0.0
12	primary_replaced__material_id	10000.0	0.0	0.0
13	gross_claim_amount	141891.0	0.0	0.0
14	gross_labor_amount	130643.0	0.0	0.0
15	gross_material_amount	106474.0	0.0	0.0
16	gross_other_amount	35288.0	0.0	0.0
17	product_unit_id	30679.0	0.0	0.0
18	repairing_state_province	57.0	0.0	0.0
19	repairing_region	5.0	0.0	0.0
20	repairing_country	2.0	0.0	0.0
21	repairing_dealer	1000.0	0.0	0.0
22	latitude	55.0	0.0	0.0
23	longitude	57.0	0.0	0.0
24	claim_repair_start_date	1190.0	0.0	0.0
25	claim_repair_end_date	1196.0	0.0	0.0
26	claim_processed_date	1215.0	0.0	0.0
27	claim_submitted_date	1199.0	0.0	0.0
28	service_year_date	1111.0	63136.0	0.0

A SASDataFrame is a subclass of pandas DataFrame. Therefore, anything you can do with a pandas DataFrame will also work with SASDataFrame. SASDataFrames are local on the client.

Create the **pctDistinct** and **pctMissing** columns using traditional pandas code.

In [52]:

```
## Number of rows in the CAS table
nRows = castbl.shape[0]

## Pandas code on the Python client to prepare the SASDataFrame
distinct_df = (distinct_df
               .assign(pctDistinct = distinct_df.NDistinct / nRows,
                       pctMissing = distinct_df.NMiss / nRows)
               .sort_values('pctDistinct', ascending = False)
               .drop('Trunc', axis = 1)
              )

distinct_df
```

Out[52]:

Distinct Counts for WARRANTY_DEMO

	Column	NDistinct	NMiss	pctDistinct	pctMissing
0	warranty_claim_id	144681.0	0.0	0.944288	0.000000
13	gross_claim_amount	141891.0	0.0	0.926079	0.000000
14	gross_labor_amount	130643.0	0.0	0.852666	0.000000
15	gross_material_amount	106474.0	0.0	0.694923	0.000000
16	gross_other_amount	35288.0	0.0	0.230314	0.000000
17	product_unit_id	30679.0	0.0	0.200232	0.000000
12	primary_replaced__material_id	10000.0	0.0	0.065267	0.000000
26	claim_processed_date	1215.0	0.0	0.007930	0.000000
27	claim_submitted_date	1199.0	0.0	0.007826	0.000000
25	claim_repair_end_date	1196.0	0.0	0.007806	0.000000
24	claim_repair_start_date	1190.0	0.0	0.007767	0.000000
28	service_year_date	1111.0	63136.0	0.007251	0.412069
21	repairing_dealer	1000.0	0.0	0.006527	0.000000
11	defect_key	745.0	0.0	0.004862	0.000000
10	primary_labor_code	80.0	0.0	0.000522	0.000000
9	primary_labor_description	80.0	0.0	0.000522	0.000000
5	trim_level	60.0	0.0	0.000392	0.000000
18	repairing_state_province	57.0	0.0	0.000372	0.000000
23	longitude	57.0	0.0	0.000372	0.000000
22	latitude	55.0	0.0	0.000359	0.000000
2	product_model	21.0	0.0	0.000137	0.000000
8	primary_labor_group	8.0	0.0	0.000052	0.000000
3	model_year	5.0	0.0	0.000033	0.000000
19	repairing_region	5.0	0.0	0.000033	0.000000
6	engine_model	4.0	0.0	0.000026	0.000000
1	make	4.0	0.0	0.000026	0.000000
7	vehicle_assembly_plant	3.0	0.0	0.000020	0.000000
4	platform	2.0	0.0	0.000013	0.000000
20	repairing_country	2.0	0.0	0.000013	0.000000

Plot the percentage of missing and distinct values using pandas on the client.

In [53]:

```
##
## Create the dataframes for the pctMissing and pctDistinct
##
```

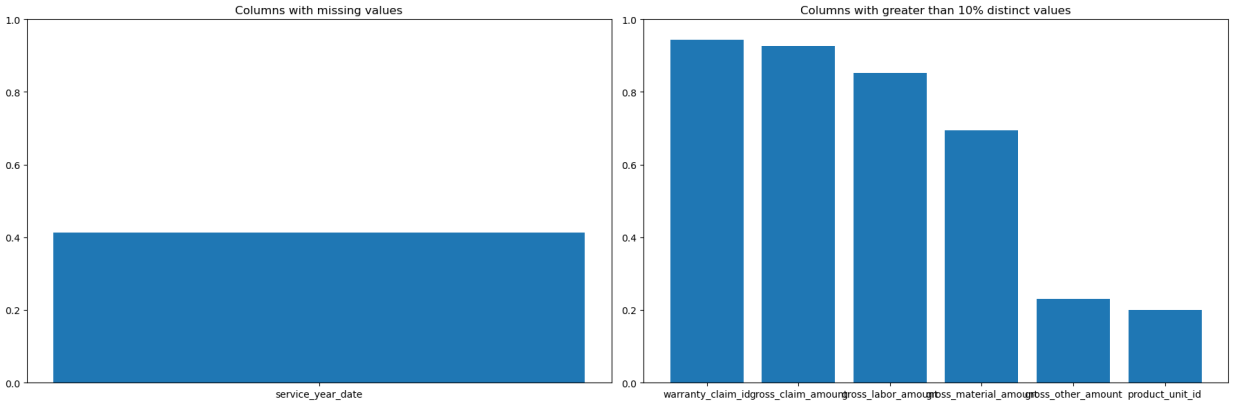
```
## Missing columns dataframe
missing = (distinct_df
           .query('pctMissing > 0')
           .loc[:,['Column', 'pctMissing']]
           .set_index('Column')
          )

## Distinct columns over 10% distinct values dataframe
distinct_gt10 = (distinct_df
                 .query('pctDistinct > .1')
                 .loc[:,['Column','pctDistinct']]
                 .set_index('Column')
                )

display(missing, distinct_gt10)


##
## Plot the dataframes
##

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize = (18,6))

ax1.bar(missing.index, missing.pctMissing)
ax1.set_title('Columns with missing values')
ax1.set_ylim(top=1)

ax2.bar(distinct_gt10.index, distinct_gt10.pctDistinct)
ax2.set_title('Columns with greater than 10% distinct values')
ax2.set_ylim(top=1)

fig.tight_layout()
```

Distinct Counts for WARRANTY_DEMO

Column	pctMissing
service_year_date	0.412069

Distinct Counts for WARRANTY_DEMO

Column	pctDistinct
warranty_claim_id	0.944288
gross_claim_amount	0.926079
gross_labor_amount	0.852666
gross_material_amount	0.694923
gross_other_amount	0.230314
product_unit_id	0.200232



b. Categorical columns

The SWAT packages contains a variety of Pandas API methods.

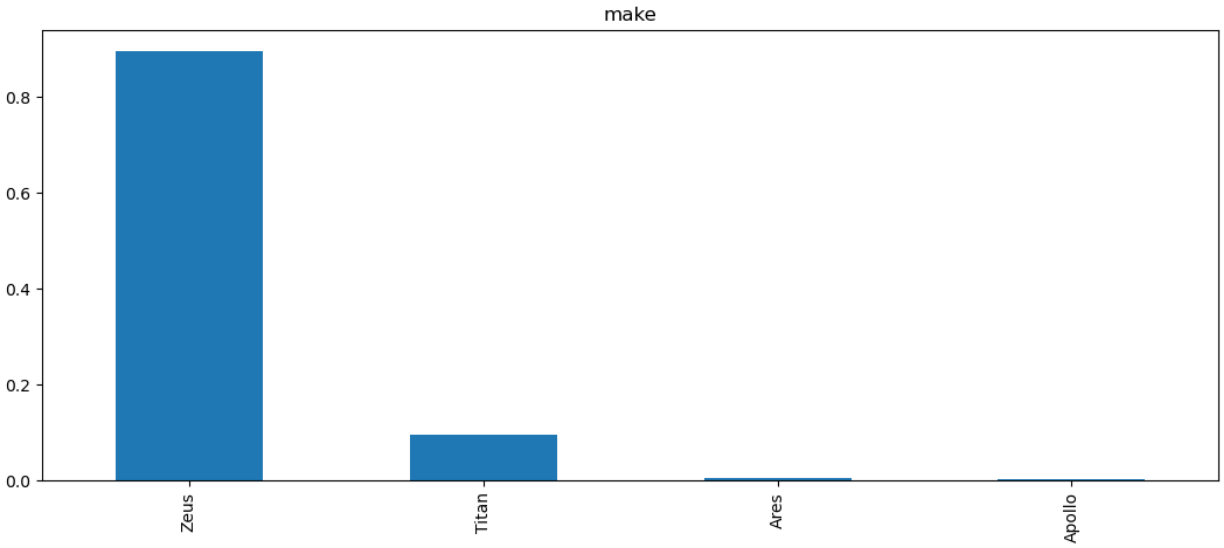
Use the value_counts method to return a Series containing the frequency of each distinct row in the CAS table.

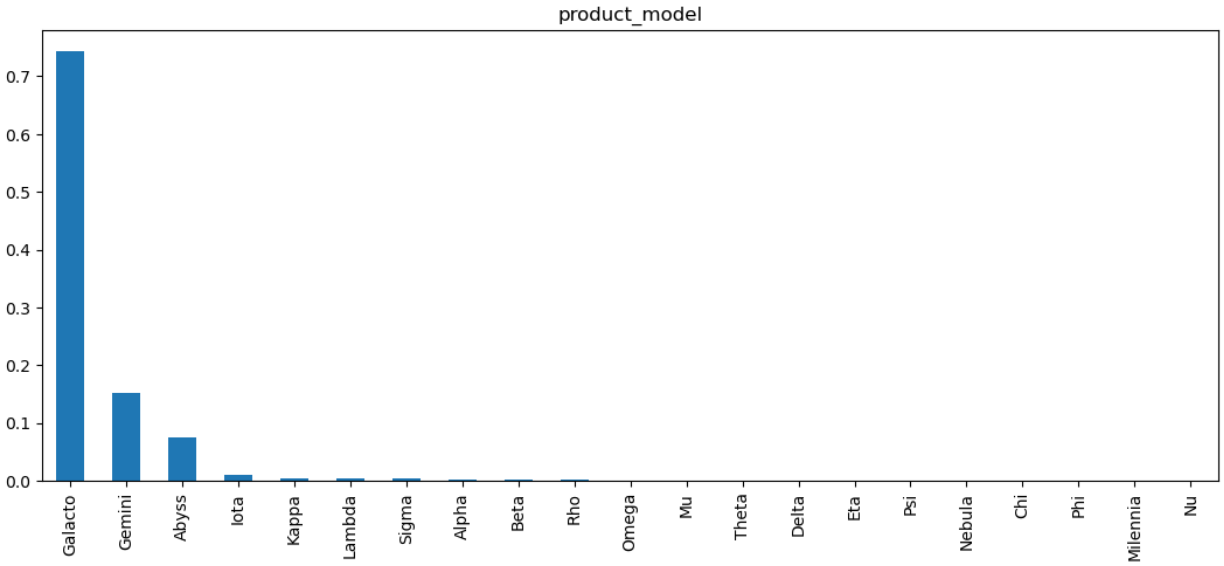
In [54]:
```
(castbl                                  ## CAS table
 .make                                   ## CAS table column
 .value_counts(normalize = True)         ## value_count swat method
)
```

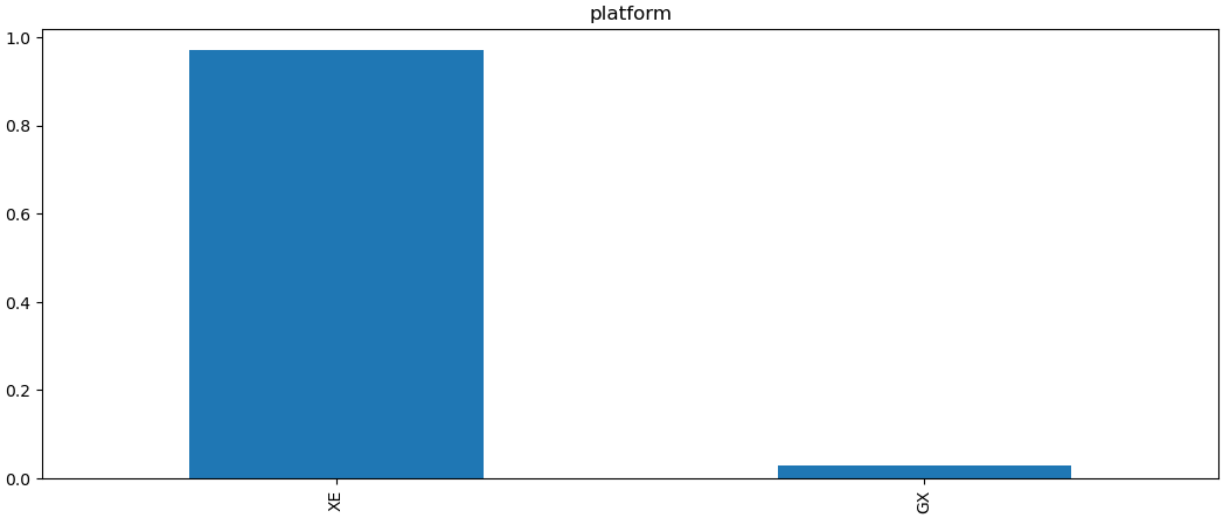
Out[54]:
```
Zeus      0.895221
Titan     0.095185
Ares      0.005783
Apollo    0.003812
dtype: float64
```

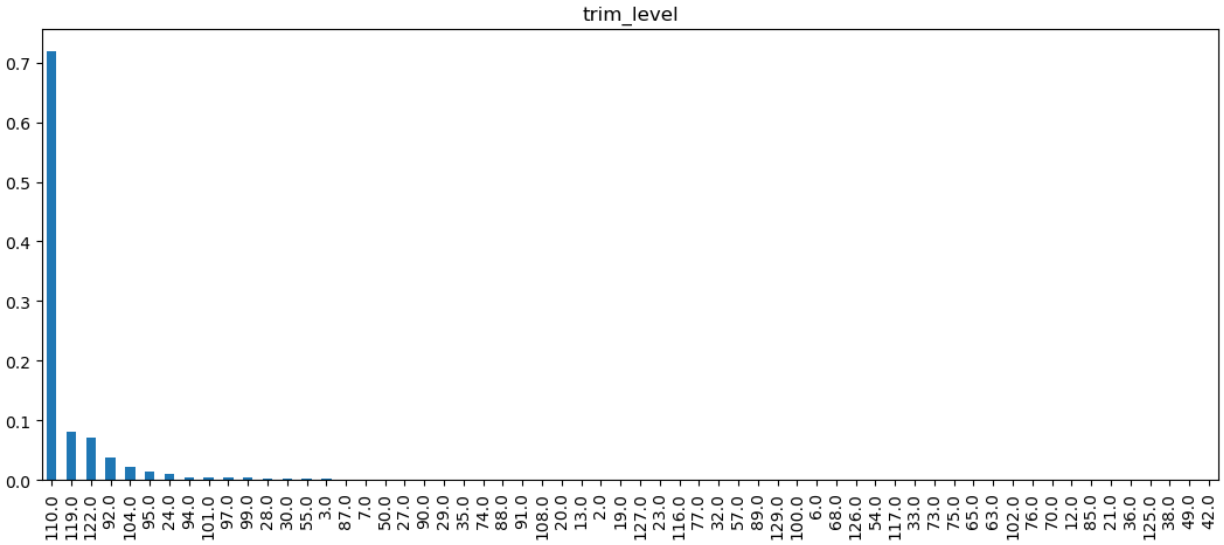
Get unique value counts of each of the following CAS table columns.

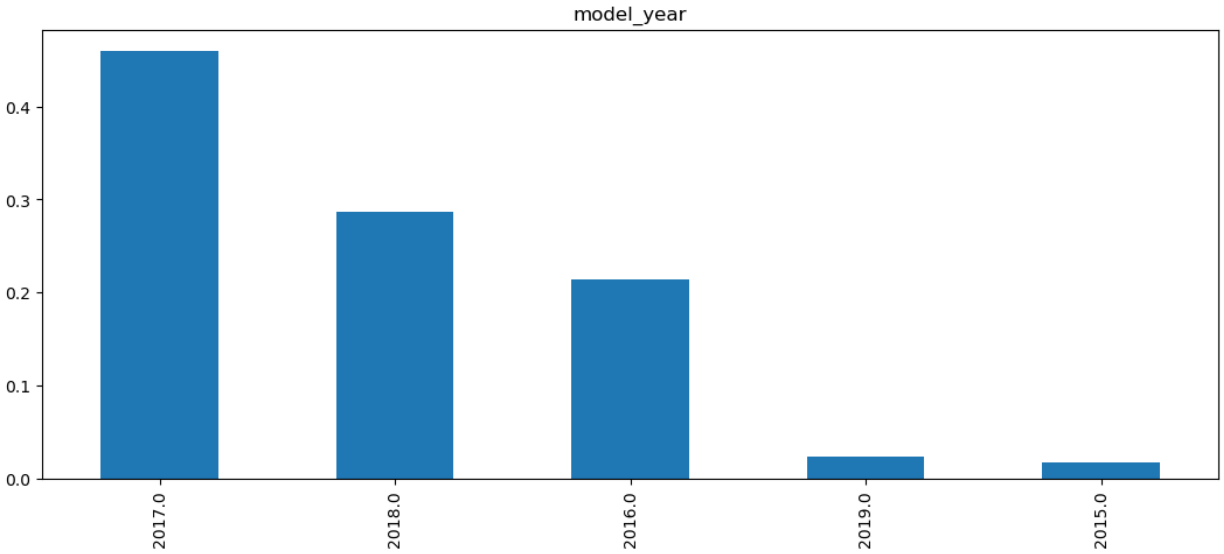
In [55]:
```
catColumns = ['make', 'product_model', 'platform', 'trim_level', 'model_year', 'engine

for col in catColumns:
    (castbl[col]                                          ## CAS table column
     .value_counts(normalize = True)                      ## value_count swat method
     .plot(kind = 'bar', title = col, figsize = (13,5))   ## Plot the SASDataFrame c
    )
    plt.show()
```

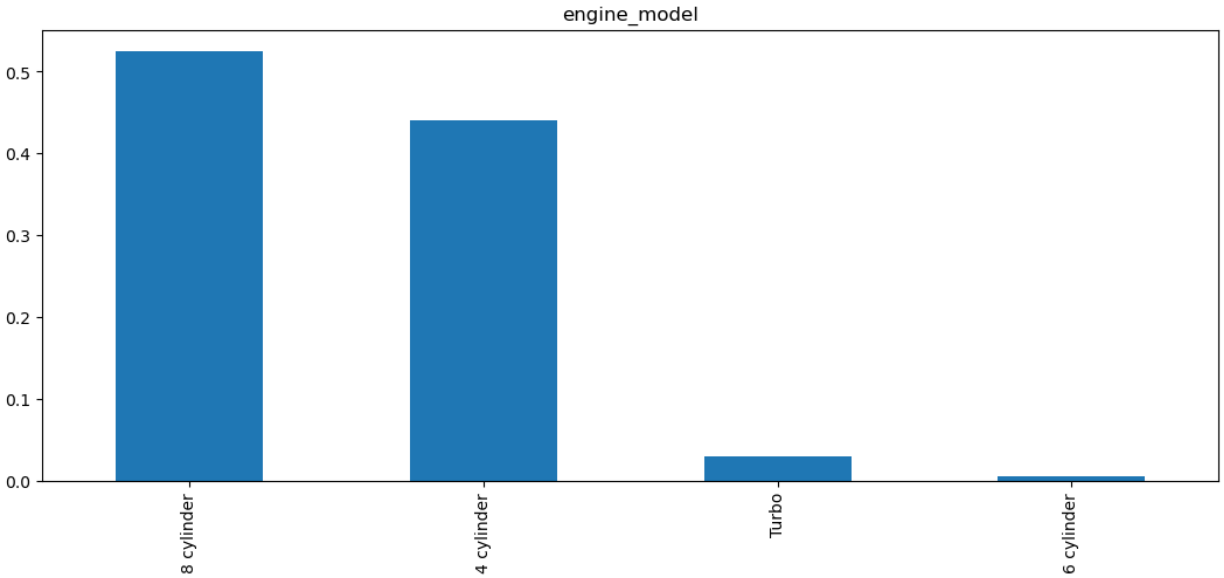

make
0.0
0.2
0.4
0.6
0.8
Zeus
Titan
Ares
Apollo

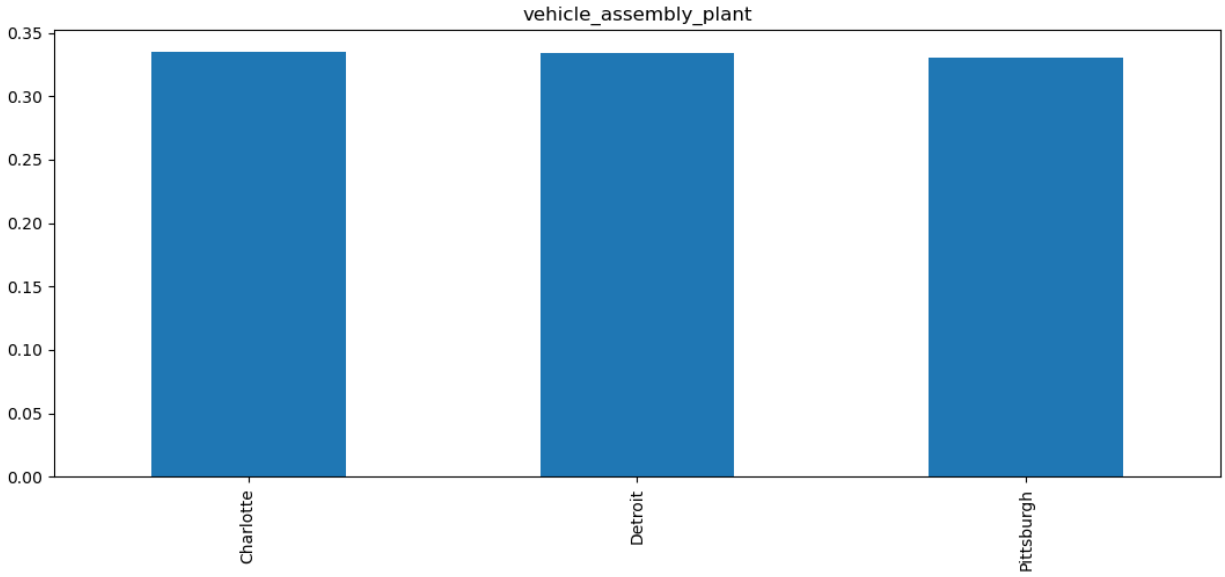

product_model
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Galacto
Gemini
Abyss
Iota
Kappa
Lambda
Sigma
Alpha
Beta
Rho
Omega
Mu
Theta
Delta
Eta
Psi
Nebula
Chi
Phi
Milennia
Nu

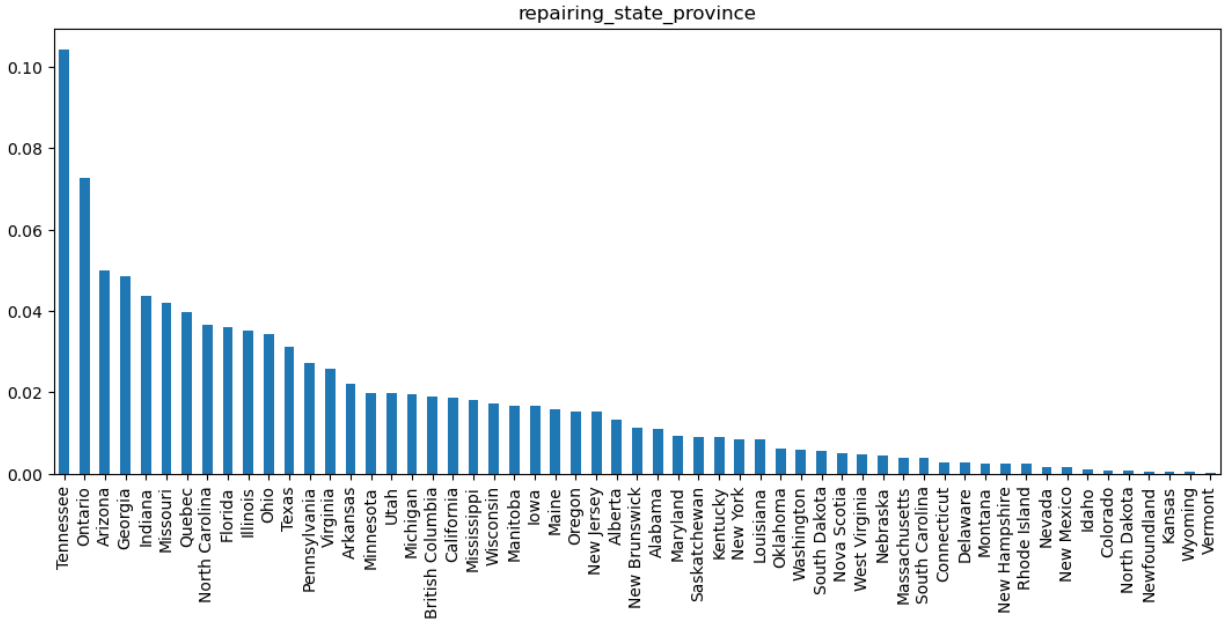

platform
0.0
0.2
0.4
0.6
0.8
1.0
XE
GX

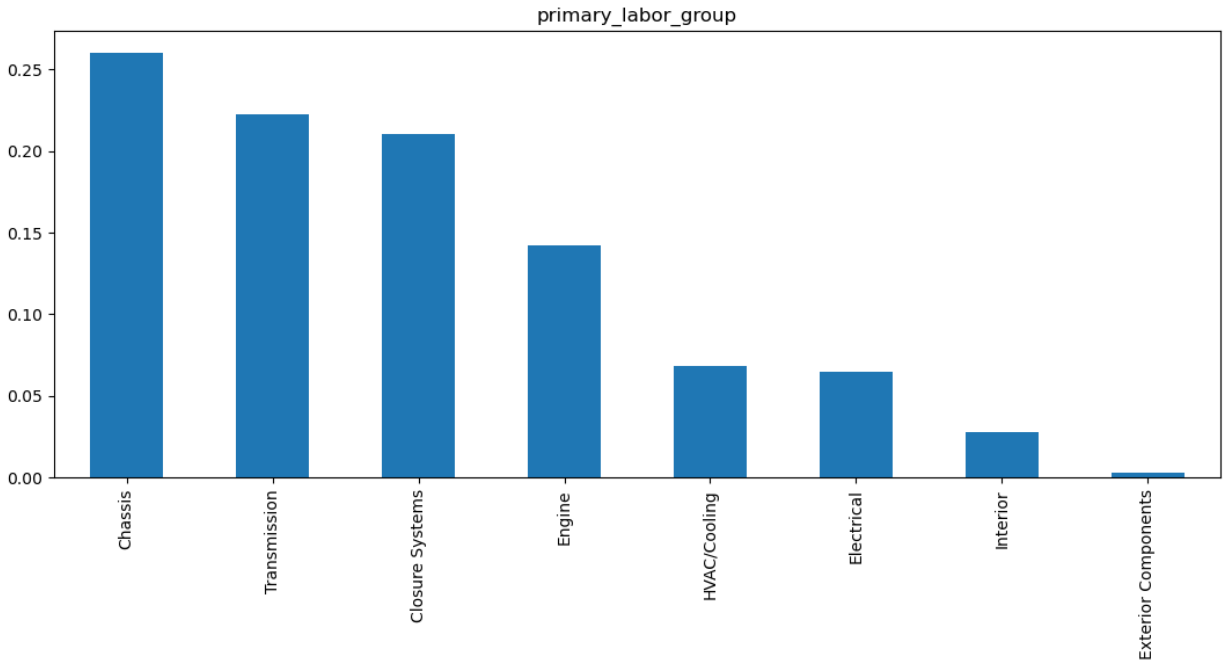

trim_level
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
110.0
119.0
122.0
92.0
104.0
95.0
24.0
94.0
101.0
97.0
99.0
28.0
30.0
55.0
3.0
87.0
7.0
50.0
27.0
90.0
29.0
35.0
74.0
88.0
91.0
108.0
20.0
13.0
2.0
19.0
127.0
23.0
116.0
77.0
32.0
57.0
89.0
129.0
100.0
6.0
68.0
126.0
54.0
117.0
33.0
73.0
75.0
65.0
63.0
102.0
76.0
70.0
12.0
85.0
21.0
36.0
125.0
38.0
49.0
42.0


model_year
0.4
0.3
0.2
0.1
0.0
2017.0
2018.0
2016.0
2019.0
2015.0


engine_model
0.5
0.4
0.3
0.2
0.1
0.0
8 cylinder
4 cylinder
Turbo
6 cylinder


vehicle_assembly_plant
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Charlotte
Detroit
Pittsburgh


repairing_state_province
0.10
0.08
0.06
0.04
0.02
0.00
Tennessee
Ontario
Arizona
Georgia
Indiana
Missouri
Quebec
North Carolina
Florida
Illinois
Ohio
Texas
Pennsylvania
Virginia
Arkansas
Minnesota
Utah
Michigan
British Columbia
California
Mississippi
Wisconsin
Manitoba
Iowa
Maine
Oregon
New Jersey
Alberta
New Brunswick
Alabama
Maryland
Saskatchewan
Kentucky
New York
Louisiana
Oklahoma
Washington
South Dakota
Nova Scotia
West Virginia
Nebraska
Massachusetts
South Carolina
Connecticut
Delaware
Montana
New Hampshire
Rhode Island
Nevada
New Mexico
Idaho
Colorado
North Dakota
Newfoundland
Kansas
Wyoming
Vermont


primary_labor_group
0.25
0.20
0.15
0.10
0.05
0.00
Chassis
Transmission
Closure Systems
Engine
HVAC/Cooling
Electrical
Interior
Exterior Components

Pull information out of the **primary_labor_description** column. View the detailed labor descriptions of the top 20 claims.

In [56]:
```
(castbl                                ## CAS table
 .primary_labor_description            ## CAS table column
 .value_counts(normalize = True)       ## SWAT value_counts method to process data on th
 .iloc[:20]                            ## Subset the Series on the client using pandas
)
```

Out[56]:
```
Cable: Cruise Control - Replace                    0.046379
Seal/Boot: Floor Gearshift - Replace               0.028313
Battery - Replace                                  0.028169
Lever: Man Trns Shift Equalzr                      0.028084
Plate: Floor Gearshift Trim - Replace              0.027836
Cable: Shift - Replace                             0.027823
Lever: Gearshift Ctl - Replace                     0.027725
Knob: Gearshift Lever - Repair/Replace             0.027699
Flywheel: Automatic Trans - Replace                0.027530
Linkage: Shift - Adjust                            0.027530
Strut: Both Rear - Replace                         0.026492
Bushings: Rr Suspn Crossmember - Replace           0.026414
Actuator: Rt Rear Select Ride - Replace            0.026335
Damper: Rear Brake Vibration - Repair/Replace      0.026139
Mount: Rear Axle Cradle Right - Replace            0.026113
Crossmember Asm: Rr Suspension - Repair/Replace    0.026113
Dampening Package Right - Install                  0.025983
Module: Four Whl Anti Lock - Replace               0.025643
Mount Assembly: B-Rr Strut Brg - Replace           0.025565
Shock Absorber:R-Super Lift - Replace              0.025526
dtype: float64
```

View the CASTable object.

In [57]:
```
castbl
```

Out[57]:
```
CASTable('WARRANTY_DEMO', caslib='casuser')
```

Add parameters to a CASTable object to calculate a new column. Notice the output adds how to calculate the new column to the CASTable object. No column has been created.

The eval method enables you to use SAS functions and assignment statements to create columns.

In [58]:
```
castbl.eval("primary_labor_item = scan(primary_labor_description,1,'-:-','r')", inplac
```

Out[58]:
```
{'name': 'WARRANTY_DEMO',
 'caslib': 'casuser',
 'computedvars': ['primary_labor_item'],
 'computedvarsprogram': "primary_labor_item =  scan(primary_labor_description,1,'-:-
','r'); primary_labor_item = primary_labor_item; "}
```

View the original CASTable object.

In [59]:
```
castbl
```

Out[59]:
```
CASTable('WARRANTY_DEMO', caslib='casuser')
```

Create a new column in the CAS table and summarizes the unique counts.

In [60]:
```
(castbl
 .eval("primary_labor_item = scan(primary_labor_description,1,'-:-','r')", inplace = F
 .primary_labor_item
 .value_counts(normalize = True)
)
```

Out[60]:

Cable	0.074202
Latch	0.062748
Lever	0.055810
Module	0.046209
Seal	0.042189
Relay	0.040818
Plate	0.030597
Seal/Boot	0.028313
Battery	0.028169
Knob	0.027699
Flywheel	0.027530
Linkage	0.027530
Strut	0.026492
Bushings	0.026414
Actuator	0.026335
Damper	0.026139
Crossmember Asm	0.026113
Mount	0.026113
Dampening Package Right	0.025983
Mount Assembly	0.025565
Shock Absorber	0.025526
Spring	0.021877
Lock & Bolt Assembly	0.021016
Striker	0.021016
Support	0.021003
Secondary Ltch Hndl/Rod	0.020859
Valve	0.019025
Pump & Motor Assy	0.009901
Governor	0.009862
Hose	0.009725
Electronic Control Unit (ECU)	0.009614
Energy Unit Assembly	0.009562
Booster	0.009464
Compressor	0.009150
Sensor	0.007382
Rear A/C Blower Motor	0.007297
Capacitor	0.007075
Switch	0.007003
Hvac Performance Check	0.006709
Diode	0.003936
Grille/Lamp Sensor	0.003759
Controller	0.003629
Resistor	0.003531
Bulbs	0.003485
Molding	0.003022
Reflector	0.002898
Emblem/Namplt	0.002865
Console	0.002800
Carpet/Pad	0.002741
Tray	0.002709
Trim	0.002689
Plate/Slide	0.002682
Cup Holder	0.002656
Antenna	0.000842
Glass	0.000672
Spoiler	0.000372
Cowl Area	0.000352
Emblem/Base	0.000326

dtype: float64

View the CASTable object. Notice it still references the CAS table with no additional parameters.

In [61]:
```
castbl
```

Out[61]:
```
CASTable('WARRANTY_DEMO', caslib='casuser')
```

c. Numeric columns

Generate descriptive statistics of a CAS table using the Pandas API from the SWAT package.

In [62]:
```
numericColumns = ['gross_claim_amount', 'gross_labor_amount', 'gross_material_amount',

(castbl                          ## CAS table
 .loc[:,numericColumns]          ## CAS table columns
 .describe()                     ## swat describe method, processes in the CAS server
)
```

Out[62]:

	gross_claim_amount	gross_labor_amount	gross_material_amount	gross_other_amount
count	153217.000000	153217.000000	153217.000000	153217.000000
mean	540.225731	184.149760	330.067235	26.008735
std	1375.949840	288.125026	1241.841830	161.888485
min	0.000000	0.000000	0.000000	0.000000
25%	111.988329	62.366981	0.000000	0.000000
50%	208.895870	118.853158	39.035339	0.000000
75%	439.881829	214.006799	168.183010	0.000000
max	75292.757377	14134.469857	70035.052932	16688.276805

d. Date columns

Calculate the **days_to_repair** and **days_to_process_claim** columns.

In [63]:
```
(castbl
 .eval("days_to_repair = claim_repair_end_date - claim_repair_start_date", inplace = T
 .eval("days_to_process_claim = claim_processed_date - claim_submitted_date", inplace
 .loc[:, ['days_to_repair', 'days_to_process_claim']]
 .describe()
)
```

Out[63]:

	days_to_repair	days_to_process_claim
count	153217.000000	153217.000000
mean	3.043527	4.553176
std	0.542297	2.925297
min	0.000000	0.000000
25%	3.000000	2.000000
50%	3.000000	4.000000
75%	3.000000	7.000000
max	10.000000	11.000000

5. Prepare final data

a. Using Python

Confirm the castbl variable references the CAS WARRANTY_DEMO CAS table.

In [64]:
```
castbl
```

Out[64]:
```
CASTable('WARRANTY_DEMO', caslib='casuser')
```

In [65]:
```
castbl.tableInfo()
```

Out[65]:

§ TableInfo

	Name	Rows	Columns	IndexedColumns	Encoding	CreateTimeFormatted	ModTimeF
0	WARRANTY_DEMO	153217	29	0	utf-8	2023-09-19T13:42:06+00:00	19T13:43

elapsed 0.00748s · user 0.00638s · sys 0.00696s · mem 1.88MB

NOTE: If you are using SAS Viya for Learners 3.5 you must use the partition action instead of copyTable.

SAS Viya Create the following columns in the CAS table **WARRANTY_FINAL**:

- **days_to_repair**
- **days_to_process_claim**
- **primary_labor_item**
- **car_serviced**

In [66]:
```
(castbl
 .eval("days_to_repair = claim_repair_end_date - claim_repair_start_date", inplace = F
 .eval("days_to_process_claim = claim_processed_date - claim_submitted_date", inplace
 .eval("primary_labor_item = scan(primary_labor_description,1,'-:-','r')", inplace = F
```

```
 .eval("car_serviced = ifc(service_year_date = ., 'Not Serviced', 'Serviced')", inplac
 .copyTable(casout = {'name':'warranty_final',
                      'caslib':'casuser',
                      'replace':True})
)
```

Out[66]:

§ caslib

CASUSER(Peter.Styliadis@sas.com)

---

§ tableName

WARRANTY_FINAL

---

§ casTable

CASTable('WARRANTY_FINAL', caslib='CASUSER(Peter.Styliadis@sas.com)')

elapsed 0.145s · user 0.842s · sys 0.736s · mem 747MB

SAS Viya for Learners 3.5

In [59]:

```
##############################################################################################
## Use the following code if you are using SAS Viya for Learners (Viya 3.5)
## In SAS Viya 3.5 you must use the partition action instead of copyTable. Code is sim
##############################################################################################
# (castbl
#  .eval("days_to_repair = claim_repair_end_date - claim_repair_start_date", inplace =
#  .eval("days_to_process_claim = claim_processed_date - claim_submitted_date", inplac
#  .eval("primary_labor_item = scan(primary_labor_description,1,'-:-','r')", inplace =
#  .eval("car_serviced = ifc(service_year_date = ., 'Not Serviced', 'Serviced')", inpl
#  .partition(casout = {'name':'warranty_final', 'caslib':'casuser', 'replace':True})
# )
```

Confirm the **WARRANTY_FINAL** CAS table was created.

In [67]:

```
conn.tableInfo(caslib = 'casuser')
```

Out[67]:

§ TableInfo

	Name	Rows	Columns	IndexedColumns	Encoding	CreateTimeFormatted	ModTimeF
0	WARRANTY_DEMO	153217	29	0	utf-8	2023-09-19T13:42:06+00:00	19T13:43
1	WARRANTY_FINAL	153217	33	0	utf-8	2023-09-19T13:44:35+00:00	19T13:44

elapsed 0.00779s · user 0.0102s · sys 0.00258s · mem 1.88MB

Reference the new **WARRANTY_FINAL** CAS table.

In [68]:

```
finalTbl = conn.CASTable('WARRANTY_FINAL', caslib = 'casuser')
finalTbl.head()
```

Out[68]:

	warranty_claim_id	make	product_model	model_year	platform	trim_level	engine_
0	C1_00797404_4V4NC9TG54N356290	Zeus	Galacto	2017.0	XE	110.0	4 c
1	C1_00803803_4V4NC9GH64N355670	Zeus	Galacto	2017.0	XE	110.0	8 c
2	C1_50938501_4V4NC9GF02N337502	Zeus	Galacto	2015.0	XE	110.0	8 c
3	C1_51422203_4V4NC9GH92N337872	Zeus	Galacto	2015.0	XE	110.0	8 c
4	C1_51560302_4V4NC9TJ62N337871	Zeus	Galacto	2015.0	XE	110.0	4 c

b. Using SQL

Load the fedSQL action set.

In [69]:
```
conn.loadActionSet('fedSQL')
```

```
NOTE: Added action set 'fedSQL'.
```

Out[69]:

§ actionset

fedSQL

elapsed 0.0104s · user 0.0109s · sys 0.012s · mem 1.25MB

View available actions in the fedSQL action set.

In [70]:
```
conn.fedSQL?
```

```
Signature:   conn.fedSQL(*args, **kwargs)
Type:        Fedsql
String form: <swat.cas.actions.Fedsql object at 0x000001DE9A0213A0>
File:        c:\users\pestyl\anaconda3\lib\site-packages\swat\cas\actions.py
Docstring:
FedSQL

Actions
-------
fedsql.execdirect : Submits a SAS FedSQL language statement for immediate
                    execution
```

Use SQL to prepare the CAS table. The results are similar to the Python technique.

In [71]:
```
createTableQuery = '''
    create table casuser.warranty_final_sql{options replace=True} as
    select *,
           claim_repair_end_date - claim_repair_start_date as days_to_repair,
           claim_processed_date - claim_submitted_date as days_to_process_claim,
           scan(primary_labor_description,1,'-:-','r') as primary_labor_item,
           case
               when service_year_date = . then 'Not Serviced'
               else 'Serviced'
           end as car_serviced
```

```
    from casuser.warranty_demo;
'''

conn.execdirect(query = createTableQuery)
```

```
NOTE: CASDAL driver. Creation of a DATE column has been requested, but is not support
ed by the CASDAL driver. A DOUBLE PRECISION column will be created instead. A DATE fo
rmat will be associated with the column.
NOTE: CASDAL driver. Creation of a DATE column has been requested, but is not support
ed by the CASDAL driver. A DOUBLE PRECISION column will be created instead. A DATE fo
rmat will be associated with the column.
NOTE: CASDAL driver. Creation of a DATE column has been requested, but is not support
ed by the CASDAL driver. A DOUBLE PRECISION column will be created instead. A DATE fo
rmat will be associated with the column.
NOTE: CASDAL driver. Creation of a DATE column has been requested, but is not support
ed by the CASDAL driver. A DOUBLE PRECISION column will be created instead. A DATE fo
rmat will be associated with the column.
NOTE: CASDAL driver. Creation of a DATE column has been requested, but is not support
ed by the CASDAL driver. A DOUBLE PRECISION column will be created instead. A DATE fo
rmat will be associated with the column.
NOTE: CASDAL driver. Creation of an NVARCHAR column has been requested, but is not su
pported by the CASDAL driver. A VARCHAR column will be created instead.
NOTE: Table WARRANTY_FINAL_SQL was created in caslib CASUSER(Peter.Styliadis@sas.com)
with 153217 rows returned.
```

Out[71]: elapsed 0.263s · user 2.23s · sys 1.47s · mem 797MB

Confirm the new **WARRANTY_FINAL_SQL** CAS table was created.

In [72]:
```
conn.tableInfo(caslib = 'casuser')
```

Out[72]: **§ TableInfo**

	Name	Rows	Columns	IndexedColumns	Encoding	CreateTimeFormatted	ModTi
0	WARRANTY_DEMO	153217	29	0	utf-8	2023-09-19T13:42:06+00:00	19T1
1	WARRANTY_FINAL	153217	33	0	utf-8	2023-09-19T13:44:35+00:00	19T1
2	WARRANTY_FINAL_SQL	153217	33	0	utf-8	2023-09-19T13:44:58+00:00	19T1

elapsed 0.00946s · user 0.00487s · sys 0.0105s · mem 1.9MB

c. Add column labels for the dashboard for a production report

Take the current column name and:

- replace the _ with a space
- make everything title case
- replace *Id* with *ID*

View the column names and labels.

In [73]:
```
finalTbl.columnInfo()
```

Out[73]:

§ ColumnInfo

	Column	Label	ID	Type	RawLength	FormattedLength	Format	NFL	NFD
0	warranty_claim_id		1	varchar	29	29		0	0
1	make		2	varchar	6	6		0	0
2	product_model		3	varchar	8	8		0	0
3	model_year		4	double	8	12		0	0
4	platform		5	varchar	2	2		0	0
5	trim_level		6	double	8	12		0	0
6	engine_model		7	varchar	10	10		0	0
7	vehicle_assembly_plant		8	varchar	10	10		0	0
8	primary_labor_group		9	varchar	19	19		0	0
9	primary_labor_description		10	varchar	50	50		0	0
10	primary_labor_code		11	varchar	5	5		0	0
11	defect_key		12	double	8	12		0	0
12	primary_replaced__material_id		13	varchar	5	5		0	0
13	gross_claim_amount		14	double	8	12		0	0
14	gross_labor_amount		15	double	8	12		0	0
15	gross_material_amount		16	double	8	12		0	0
16	gross_other_amount		17	double	8	12		0	0
17	product_unit_id		18	varchar	17	17		0	0
18	repairing_state_province		19	varchar	16	16		0	0
19	repairing_region		20	double	8	12		0	0
20	repairing_country		21	varchar	6	6		0	0
21	repairing_dealer		22	varchar	4	4		0	0
22	latitude		23	double	8	12		0	0
23	longitude		24	double	8	12		0	0
24	claim_repair_start_date		25	double	8	12	DATE	9	0
25	claim_repair_end_date		26	double	8	12	DATE	9	0
26	claim_processed_date		27	double	8	12	DATE	9	0
27	claim_submitted_date		28	double	8	12	DATE	9	0
28	service_year_date		29	double	8	12	DATE	9	0
29	days_to_repair		30	double	8	12		0	0
30	days_to_process_claim		31	double	8	12		0	0
31	primary_labor_item		32	char	50	50		0	0
32	car_serviced		33	char	200	200		0	0

elapsed 0.0134s · user 0.0097s · sys 0.0106s · mem 2.53MB

Create a list of dictionaries to add column labels.

In [74]:
```
addLabelsToColumns = [{'name':colname, 'label':colname.replace('_',' ').title().replac
addLabelsToColumns
```

Out[74]:
```
[{'name': 'warranty_claim_id', 'label': 'Warranty Claim ID'},
 {'name': 'make', 'label': 'Make'},
 {'name': 'product_model', 'label': 'Product Model'},
 {'name': 'model_year', 'label': 'Model Year'},
 {'name': 'platform', 'label': 'Platform'},
 {'name': 'trim_level', 'label': 'Trim Level'},
 {'name': 'engine_model', 'label': 'Engine Model'},
 {'name': 'vehicle_assembly_plant', 'label': 'Vehicle Assembly Plant'},
 {'name': 'primary_labor_group', 'label': 'Primary Labor Group'},
 {'name': 'primary_labor_description', 'label': 'Primary Labor Description'},
 {'name': 'primary_labor_code', 'label': 'Primary Labor Code'},
 {'name': 'defect_key', 'label': 'Defect Key'},
 {'name': 'primary_replaced__material_id',
  'label': 'Primary Replaced  Material ID'},
 {'name': 'gross_claim_amount', 'label': 'Gross Claim Amount'},
 {'name': 'gross_labor_amount', 'label': 'Gross Labor Amount'},
 {'name': 'gross_material_amount', 'label': 'Gross Material Amount'},
 {'name': 'gross_other_amount', 'label': 'Gross Other Amount'},
 {'name': 'product_unit_id', 'label': 'Product Unit ID'},
 {'name': 'repairing_state_province', 'label': 'Repairing State Province'},
 {'name': 'repairing_region', 'label': 'Repairing Region'},
 {'name': 'repairing_country', 'label': 'Repairing Country'},
 {'name': 'repairing_dealer', 'label': 'Repairing Dealer'},
 {'name': 'latitude', 'label': 'Latitude'},
 {'name': 'longitude', 'label': 'Longitude'},
 {'name': 'claim_repair_start_date', 'label': 'Claim Repair Start Date'},
 {'name': 'claim_repair_end_date', 'label': 'Claim Repair End Date'},
 {'name': 'claim_processed_date', 'label': 'Claim Processed Date'},
 {'name': 'claim_submitted_date', 'label': 'Claim Submitted Date'},
 {'name': 'service_year_date', 'label': 'Service Year Date'},
 {'name': 'days_to_repair', 'label': 'Days To Repair'},
 {'name': 'days_to_process_claim', 'label': 'Days To Process Claim'},
 {'name': 'primary_labor_item', 'label': 'Primary Labor Item'},
 {'name': 'car_serviced', 'label': 'Car Serviced'}]
```

Add labels to the columns.

In [75]:
```
finalTbl.alterTable(columns = addLabelsToColumns)
finalTbl.columnInfo()
```

Out[75]:

§ ColumnInfo

	Column	Label	ID	Type	RawLength	FormattedLength	Format	NFL
0	warranty_claim_id	Warranty Claim ID	1	varchar	29	29		0
1	make	Make	2	varchar	6	6		0
2	product_model	Product Model	3	varchar	8	8		0
3	model_year	Model Year	4	double	8	12		0
4	platform	Platform	5	varchar	2	2		0
5	trim_level	Trim Level	6	double	8	12		0
6	engine_model	Engine Model	7	varchar	10	10		0
7	vehicle_assembly_plant	Vehicle Assembly Plant	8	varchar	10	10		0
8	primary_labor_group	Primary Labor Group	9	varchar	19	19		0
9	primary_labor_description	Primary Labor Description	10	varchar	50	50		0
10	primary_labor_code	Primary Labor Code	11	varchar	5	5		0
11	defect_key	Defect Key	12	double	8	12		0
12	primary_replaced__material_id	Primary Replaced Material ID	13	varchar	5	5		0
13	gross_claim_amount	Gross Claim Amount	14	double	8	12		0
14	gross_labor_amount	Gross Labor Amount	15	double	8	12		0
15	gross_material_amount	Gross Material Amount	16	double	8	12		0
16	gross_other_amount	Gross Other Amount	17	double	8	12		0
17	product_unit_id	Product Unit ID	18	varchar	17	17		0
18	repairing_state_province	Repairing State Province	19	varchar	16	16		0

	Column	Label	ID	Type	RawLength	FormattedLength	Format	NFL
19	repairing_region	Repairing Region	20	double	8	12		0
20	repairing_country	Repairing Country	21	varchar	6	6		0
21	repairing_dealer	Repairing Dealer	22	varchar	4	4		0
22	latitude	Latitude	23	double	8	12		0
23	longitude	Longitude	24	double	8	12		0
24	claim_repair_start_date	Claim Repair Start Date	25	double	8	12	DATE	9
25	claim_repair_end_date	Claim Repair End Date	26	double	8	12	DATE	9
26	claim_processed_date	Claim Processed Date	27	double	8	12	DATE	9
27	claim_submitted_date	Claim Submitted Date	28	double	8	12	DATE	9
28	service_year_date	Service Year Date	29	double	8	12	DATE	9
29	days_to_repair	Days To Repair	30	double	8	12		0
30	days_to_process_claim	Days To Process Claim	31	double	8	12		0
31	primary_labor_item	Primary Labor Item	32	char	50	50		0
32	car_serviced	Car Serviced	33	char	200	200		0

elapsed 0.0108s · user 0.00279s · sys 0.0159s · mem 2.53MB

6. Create models

Load the regression action set. Here you can view all available CAS Action sets.

In [76]:
```
conn.loadActionSet('regression')
conn.regression?
```

```
NOTE: Added action set 'regression'.
```

```
Signature:   conn.regression(*args, **kwargs)
Type:        Regression
String form: <swat.cas.actions.Regression object at 0x000001DE97CBAEE0>
File:        c:\users\pestyl\anaconda3\lib\site-packages\swat\cas\actions.py
Docstring:
Regression

Actions
-------
regression.genmod              : Fits generalized linear regression models
regression.genmodscore         : creates a table on the server that contains
                                 results from scoring observations by using a
                                 fitted model
regression.glm                 : Fits linear regression models using the method
                                 of least squares
regression.glmscore            : creates a table on the server that contains
                                 results from scoring observations by using a
                                 fitted model
regression.logistic            : Fits logistic regression models
regression.logisticassociation : computes indices of rank correlation between
                                 predicted probabilities and observed responses
                                 used for assessing the predictive ability of a
                                 model
regression.logisticcode        : writes SAS DATA step code for computing
                                 predicted values of the fitted model
regression.logisticlackfit     : computes the Hosmer and Lemeshow test
regression.logisticoddsratio   : creates a table that compares subpopulations by
                                 using odds ratios.
regression.logisticscore       : creates a table on the server that contains
                                 results from scoring observations by using a
                                 fitted model
regression.logistictype3       : computes Type 3 or Joint tests that all
                                 parameters for an effect are zero
regression.modelmatrix         : creates a table on the server that contains the
                                 design matrix associated with a given model
                                 statement.
```

Fit a linear regression model using the method of least squares

In [77]:
```
cols = ['defect_key', 'trim_level','platform', 'engine_model']

cr_lr = castbl.glm(
           target = 'gross_claim_amount',
           inputs = cols,
           selection = {'method':'BACKWARD'}
)
```

View keys in the CASResults object from the glm action.

In [78]:
```
cr_lr.keys()
```

Out[78]:
```
odict_keys(['ClassInfo', 'Dimensions', 'ModelInfo', 'NObs', 'SelectedModel.ANOVA', 'S
electedModel.FitStatistics', 'SelectedModel.ParameterEstimates', 'SelectionInfo', 'Su
mmary.SelectedEffects', 'Summary.SelectionReason', 'Summary.SelectionSummary', 'Summa
ry.StopReason', 'Timing'])
```

View the results of the glm action.

In [79]:
```
cr_lr
```

Out[79]:

§ ClassInfo

Class Level Information

	Class	Levels	Values
0	platform	2.0	GX XE
1	engine_model	4.0	4 cylinder 6 cylinder 8 cylinder Turbo

---

§ Dimensions

Dimensions

	RowId	Description	Value
0	NEFFECTS	Number of Effects	5
1	NPARMS	Number of Parameters	9

---

§ ModelInfo

Model Information

	RowId	Description	Value
0	DATA	Data Source	WARRANTY_DEMO
1	RESPONSEVAR	Response Variable	gross_claim_amount

---

§ NObs

Number of Observations

	RowId	Description	Value
0	NREAD	Number of Observations Read	153217.0
1	NUSED	Number of Observations Used	153217.0

---

§ SelectedModel.ANOVA

Analysis of Variance

	RowId	Source	DF	SS	MS	FValue	ProbF
0	MODEL	Model	4.0	2.110693e+09	5.276734e+08	280.750333	6.021074e-241
1	ERROR	Error	153212.0	2.879637e+11	1.879511e+06	NaN	NaN
2	TOTAL	Corrected Total	153216.0	2.900743e+11	NaN	NaN	NaN

---

§ SelectedModel.FitStatistics

Fit Statistics

	RowId	Description	Value
0	RMSE	Root MSE	1.370953e+03
1	RSQUARE	R-Square	7.276388e-03
2	ADJRSQ	Adj R-Sq	7.250470e-03

	RowId	Description	Value
3	AIC	AIC	2.366677e+06
4	AICC	AICC	2.366677e+06
5	SBC	SBC	2.213507e+06
6	TRAIN_ASE	ASE	1.879450e+06

§ SelectedModel.ParameterEstimates

Parameter Estimates

	Effect	engine_model	Parameter	DF	Estimate	StdErr	tValue	Probt
0	Intercept		Intercept	1	849.317177	22.190223	38.274387	0.000000e+00
1	defect_key		defect_key	1	-0.277486	0.012914	-21.487202	2.906624e-102
2	engine_model	4 cylinder	engine_model 4 cylinder	1	-236.381044	21.066609	-11.220650	3.314801e-29
3	engine_model	6 cylinder	engine_model 6 cylinder	1	-282.045824	50.619190	-5.571915	2.523761e-08
4	engine_model	8 cylinder	engine_model 8 cylinder	1	-36.026326	20.947996	-1.719798	8.547114e-02
5	engine_model	Turbo	engine_model Turbo	0	0.000000	NaN	NaN	NaN

§ SelectionInfo

Selection Information

	RowId	Description	Value	NValue
0	METHOD	Selection Method	Backward	NaN
1	SELCRITERION	Select Criterion	SBC	NaN
2	STOPCRITERION	Stop Criterion	SBC	NaN
3	HIERARCHY	Effect Hierarchy Enforced	Single	NaN
4	STOPHORIZON	Stop Horizon	3	3.0

§ Summary.SelectedEffects

Selected Effects

	Label	Effects
0	Selected Effects:	Intercept defect_key engine_model

§ Summary.SelectionReason

Selection Reason

	Reason

	Reason
0	The model at step 2 is selected.

§ Summary.SelectionSummary

Selection Summary

	Control	Step	EffectRemoved	nEffectsIn	SBC	OptSBC
0		0		5	2.213528e+06	0
1	-	1	platform	4	2.213517e+06	0
2		2	trim_level	3	2.213507e+06	1
3		3	defect_key	2	2.213957e+06	0
4		4	engine_model	1	2.214579e+06	0

§ Summary.StopReason

Stop Reason

	Reason	Code
0	Backward selection stopped because all eligible effects have been dropped.	2

§ Timing

Task Timing

	RowId	Task	Time	RelTime
0	SETUP	Setup and Parsing	0.024035	0.298749
1	LEVELIZATION	Levelization	0.017419	0.216515
2	INITIALIZATION	Model Initialization	0.001632	0.020285
3	SSCP	SSCP Computation	0.029353	0.364849
4	FITTING	Model Selection	0.001523	0.018931
5	CLEANUP	Cleanup	0.005405	0.067185
6	TOTAL	Total	0.080452	1.000000

elapsed 0.0997s · user 0.629s · sys 0.683s · mem 180MB

7. Save CAS table as a data source file

Confirm the finalTbl variable references the **WARRANTY_FINAL** CAS table in the **Casuser** caslib.

In [80]:
```
finalTbl
```

Out[80]:
```
CASTable('WARRANTY_FINAL', caslib='casuser')
```

Save the CAS table as a file named **warranty_final.sashdat**.

In [81]:
```
finalTbl.save(name='warranty_final.sashdat',
              caslib = 'casuser',
              replace = True)
```

```
NOTE: Cloud Analytic Services saved the file warranty_final.sashdat in caslib CASUSER
(Peter.Styliadis@sas.com).
```

Out[81]:

§ caslib

CASUSER(Peter.Styliadis@sas.com)

---

§ name

warranty_final.sashdat

elapsed 0.302s · user 0.0334s · sys 0.249s · mem 111MB

Confirm the **warranty_final.sashdat** file was created in the **Casuser** caslib.

In [83]:
```
conn.fileInfo(caslib = 'casuser', path ='warranty_final.sashdat')
```

Out[83]:

§ FileInfo

	Permission	Owner	Group	Name	Size	Encryption	Time	Mo
0	-rwxr-xr-x	sas	sas	warranty_final.sashdat	116881352	NONE	2023-09-19T13:45:43+00:00	2.01075

elapsed 0.02s · user 0.00348s · sys 0.0123s · mem 1.84MB

8. Terminate the CAS connection

In [84]:
```
conn.terminate()
```

9. Open the **warranty_final.sashdat** in SAS Visual Analytics

1. Log into SAS Viya
2. Select show applications on the top left
3. Select Explore & Visualize
4. Select Start with Data. Navigate to the **Casuser** caslib and load the **warranty_final.sashdat** file into memory

Additional Resources

SAS Documentation

- SAS® Cloud Analytic Services: Fundamentals
- SWAT Package

- CAS Actions

SAS Courses

- SAS® Viya® and Python Integration Fundamentals
- SAS® Viya® and Python Integration for Machine Learning
- High-Performance Data Processing with CASL in SAS® Viya®

Blog Series

- Getting Started with Python Integration to SAS® Viya®
- CAS Action! - a series on fundamentals

Additional Resources

- SAS Explore 2022 - Using Python for Data Analytics in SAS Viya
- Free Webinar - Ask the Expert Webinar - How Do I Use Python in SAS® Viya®?
- YouTube Tutorial - SAS Tutorial | Python Integration with SAS Viya
- SAS Viya - Getting Started with the Python Interface of SAS Viya
- SAS Communities - Loading Data from Python into CAS
- SAS Communities - 4 Approaches for Parallel Data Loading to CAS
- SAS Paper - Seriously Serial or Perfectly Parallel Data Transfer with SAS® Viya®
- SAS Communities - Hotwire your SWAT inside SAS Studio!

Thank You for Attending!

Connect with me on LinkedIn!

Peter Styliadis Technical Training Consultant at SAS