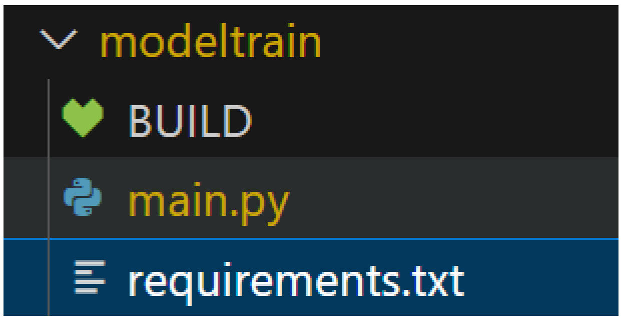
Web(REST) Clients

Python scikit-learn Example

1. Open pythontutorial folder and create modeltrain folder



2. modeltrain/BUILD

```
load("@rules_python//python:defs.bzl", "py_library", "py_binary")
load("@my_deps//:requirements.bzl", "requirement")

py_binary(
    name = "main",
    srcs = ["main.py"],
    deps = [requirement("scikit-learn")],
)
```

1. Pythontutorial/WORKSPACE

```

load("@bazel_tools//tools/build_defs/repo:http.bzl", "http_archive")

http_archive(
    name = "rules_python",
    sha256 = "c68bdc4fbec25de5b5493b8819cfc877c4ea299c0dcb15c244c5a00208cde311",
    strip_prefix = "rules_python-0.31.0",
    url = "https://github.com/bazelbuild/rules_python/releases/download/0.31.0/rules_python-0.31.0.tar.gz",
)

load("@rules_python//python:repositories.bzl", "py_repositories")

py_repositories()

load("@rules_python//python:pip.bzl", "pip_parse")
pip_parse(
    name = "my_deps",
    requirements_lock = "//modeltrain:requirements.txt",
)

load("@my_deps//:requirements.bzl", "install_deps")
install_deps()


```

4. main.py

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Train decision tree classifier
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

# Predict on test data
```

```
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

5. requirements.txt

```
scikit-learn==1.4.0
Numpy==1.26.4
scipy==1.13.0
joblib==1.4.0
threadpoolctl==3.4.0
```

6. run the target main

```
$ bazel run //modeltrain:main
```

7. create a file nltkmain in modeltrain folder

modeltrain/nltkmain.py

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist

# Download NLTK resources (if not already downloaded)
nltk.download('punkt')

# Function to analyze text and display word frequency
def analyze_text(text):
    # Tokenize the text
    words = word_tokenize(text)

    # Calculate word frequency
    freq_dist = FreqDist(words)

    # Print most common words
    print("Most common words:")
    for word, frequency in freq_dist.most_common(10):
        print(f"{word}: {frequency}")
```

```python
# Example usage
if __name__ == "__main__":
    text = "NLTK is a powerful library for natural language processing tasks. 
It provides easy-to-use interfaces for various NLP tasks."
    analyze_text(text)
```

8. modeltrain/requirements.txt

```
scikit-learn==1.4.0
Numpy==1.26.4
scipy==1.13.0
joblib==1.4.0
threadpoolctl==3.4.0
nltk==3.8.1
regex==2024.4.16
click==8.1.7
tqdm==4.66.2
colorama==0.4.6
```

9. pythontutorial/BUILD

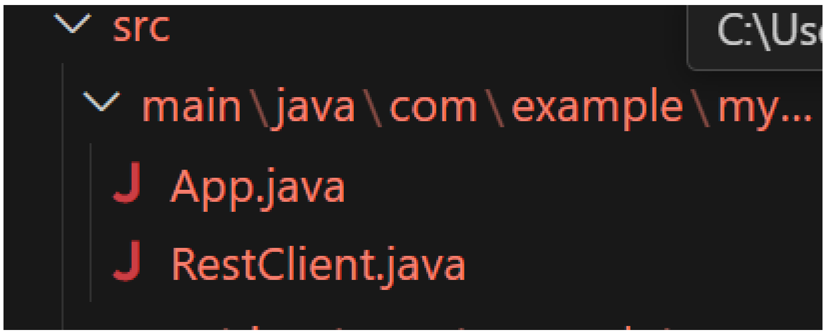
```
load("@rules_python//python:defs.bzl", "py_library", "py_binary")
load("@my_deps//:requirements.bzl", "requirement")

py_binary(
    name = "main",
    srcs = ["main.py"],
    deps = [requirement("scikit-learn")],
)
py_binary(
    name = "nltkmain",
    srcs = ["nltkmain.py"],
    deps = [requirement("nltk")],
)
```

10. run the target

$bazel run //modeltrain:nltkmain

JAVA example

1. Create a java-maven folder
2. Create WORKSAPCE and MODULE.bazel file
3. Create src/main/java/com/example/myproject/RestClient.java



4. **RestClient.java**

```java
package main.java.com.example.myproject;

import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import java.io.IOException;

public class RestClient {
    public static void main(String[] args) {
        CloseableHttpClient httpClient = HttpClients.createDefault();
        HttpGet request = new HttpGet("https://jsonplaceholder.typicode.com/posts/1");

        try (CloseableHttpResponse response = httpClient.execute(request)) {
            System.out.println("Response status: " + response.getStatusLine());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

5. java-maven/MODULE.bazel

```
bazel_dep(name = "rules_jvm_external", version = "5.3")

maven = use_extension("@rules_jvm_external//:extensions.bzl", "maven")
maven.install(
    artifacts = [
        "junit:junit:4.12",
        "com.google.guava:guava:28.0-jre",
        "org.apache.httpcomponents:httpclient:4.5.13",
    ],
    fetch_sources = True,
    repositories = [
        "https://maven.google.com",
        "https://repo1.maven.org/maven2",
        "https://jcenter.bintray.com/",
        "http://uk.maven.org/maven2",
    ],
)
use_repo(maven, "maven")
```

6. java-maven/BUILD

```
load("@rules_java//java:defs.bzl", "java_binary", "java_library", "java_test")

package(default_visibility = ["//visibility:public"])
```

```
java_library(
    name = "java-maven-lib",
    srcs = glob(["src/main/java/com/example/myproject/*.java"]),
    deps = ["@maven//:com_google_guava_guava",
             "@maven//:junit_junit"
        ],
)

java_binary(
    name = "java-maven",
    main_class = "main.java.com.example.myproject.App",
    runtime_deps = [":java-maven-lib"],
)

java_test(
    name = "tests",
    srcs = glob(["src/test/java/com/example/myproject/*.java"]),
    test_class = "test.java.com.example.myproject.TestApp",
    deps = [":java-maven-lib"],
)

java_library(
    name = "restclientlib",
    srcs = glob(["src/main/java/com/example/myproject/RestClient.java"]),
    deps = ["@maven//:org_apache_httpcomponents_httpclient"],

)

java_binary(
    name = "restclient",
    main_class = "main.java.com.example.myproject.RestClient",
    runtime_deps = [":restclientlib"],
)
```

7. run restclient target

```
$bazel run //:restclient
```

Customized Rules

Shell.bzl

```
def _emit_size_impl(ctx):
    # The input file is given to us from the BUILD file via an attribute.
    in_file = ctx.file.file

    # The output file is declared with a name based on the target's name.
    out_file = ctx.actions.declare_file("%s.size" % ctx.attr.name)
    ctx.actions.run_shell(
        # Input files visible to the action.
        inputs = [in_file],
        # Output files that must be created by the action.
        outputs = [out_file],
        # The progress message uses `short_path` (the workspace-relative path)
        # since that's most meaningful to the user. It omits details from the
        # full path that would help distinguish whether the file is a source
        # file or generated, and (if generated) what configuration it is built
        # for.
        progress_message = "Getting size of %s" % in_file.short_path,
        # The command to run. Alternatively we could use '$1', '$2', etc., and
        # pass the values for their expansion to `run_shell`'s `arguments`
        # param (see convert_to_uppercase below). This would be more robust
        # against escaping issues. Note that actions require the full `path`,
        # not the ambiguous truncated `short_path`.
        command = "wc -c '%s' | awk '{print $1}' > '%s'" %
                  (in_file.path, out_file.path),
    )

    # Tell Bazel that the files to build for this target includes
    # `out_file`.
    return [DefaultInfo(files = depset([out_file]))]

emit_size = rule(
    implementation = _emit_size_impl,
    attrs = {
        "file": attr.label(
            mandatory = True,
            allow_single_file = True,
            doc = "The file whose size is computed",
        ),
    },
    doc = """
Given an input file, creates an output file with the extension `.size`
containing the file's size in bytes.
""",
)
```

BUILD file

```
load("//:shell.bzl", "emit_size")
emit_size(
    name = "f1",
    file = "f1.txt",
)
```