(a) Optimize code structure

```
public Point2D transform(Point2D p){
  if (p == null)   return null;
  return transform.transform(p, null);
}
```

```
public Point2D transform(Point2D p) {
  return p == null ? null : transform.transform(p, null);
}
```

(b)Optimize variable name

```
public static void clearRecentFiles() {
  String str = ProgramProperties.get(RECENTFILES, "");
  if (str.length() != 0) {
    ProgramProperties.put(RECENTFILES, "");
    notifyListChange(RECENTFILES);
  }
}
```

```
public static void clearRecentFiles() {
  String recentFiles = ProgramProperties.get(RECENTFILES, "");
  if (recentFiles.length() != 0) {
    ProgramProperties.put(RECENTFILES, "");
    notifyListChange(RECENTFILES);
  }
}
```

(c)Optimize exception handling

```
public String writeDataFile() throws DataFileException {
    ByteArrayOutputStream bos = new ByteArrayOutputStream();
    writeDataFile(bos);
    String outString = bos.toString();
    try {
        if (bos != null)
            bos.close();
    } catch (IOException e) {
        Debug.logWarning(e, module);
    }
    return outString;
}
```

```
public String writeDataFile() throws DataFileException {
    try (ByteArrayOutputStream bos = new ByteArrayOutputStream()) {
        writeDataFile(bos);
        return bos.toString();
    } catch (IOException e) {
        Debug.logWarning(e, module);
        return "";
    }
}
```

(d)Delete redundant code

```
def _maybe_get_pandas_wrapper(X, trim_head=None, trim_tail=None):
    if _is_using_pandas(X, None):
        return _get_pandas_wrapper(X, trim_head, trim_tail)
    else:
        return
```

```
def _maybe_get_pandas_wrapper(X, trim_head=None, trim_tail=None):
    if _is_using_pandas(X, None):
        return _get_pandas_wrapper(X, trim_head, trim_tail)
```

(e)Complete missing code

```
def parse_time(value):
    match = time_re.match(value)
    if match:
        kw = match.groupdict()
        if kw['microsecond']:
            kw['microsecond'] = kw['microsecond'].ljust(6, '0')
        kw = dict(((k, int(v)) for (k, v) in six.iteritems(kw) if (v is not None)))
        return datetime.time(**kw)
```

```
def parse_time(value):
    time_re = re.compile(r'^(?P<hour>[0-2]?[0-9]):(?P<minute>[0-5][0-9]):(?P<second>[0-5][0-9])(?:\.(?P<microsecond>\d{1,6}))?$')
    match = time_re.match(value)
    if match:
        kw = match.groupdict()
        if kw['microsecond']:
            kw['microsecond'] = kw['microsecond'].ljust(6, '0')
        kw = dict(((k, int(v)) for (k, v) in six.iteritems(kw) if (v is not None)))
        return datetime.time(**kw)
```