execution_report.html

Report generated on 30-Nov-2022 at 22:52:43 by pytest-html v3.2.0

Environment

Packages	{"pluggy": "1.0.0", "pytest": "7.2.0"}
Platform	macOS-10.15.7-x86_64-i386-64bit
Plugins	{"html": "3.2.0", "metadata": "2.0.4"}
Python	3.9.6

Summary

86 tests ran in 251.22 seconds.

(Un)check the boxes to filter the results.

30 passed, 0 skipped, 56 failed, 0 errors, 0 expected failures, 0 unexpected passes

Results

Show all details / Hide all details

Result	Test	Duration	Links
Passed *(show details)*	test_airline_delays.py::test_year_between2015and2019_distance_lessthan1000miles	10.75	
Passed *(show details)*	test_airline_delays.py::test_year_between2015and2019_distance_between1000and4000miles	3.36	
Passed *(show details)*	test_airline_delays.py::test_year_between2015and2019_distance_morethan4000miles	3.07	
Passed *(show details)*	test_airline_delays.py::test_year_between2020and2021_distance_lessthan1000miles	1.93	
Passed *(show details)*	test_airline_delays.py::test_year_between2020and2021_distance_between1000and4000miles	2.55	
Passed *(show details)*	test_airline_delays.py::test_year_between2020and2021_distance_morethan4000miles	1.14	
Passed *(show details)*	test_airline_stats.py::test_year_between1990and1994_distance_lessthan1000miles	0.12	
Passed *(show details)*	test_airline_stats.py::test_year_between1990and1994_distance_between1000and4000miles	0.12	
Passed *(show details)*	test_airline_stats.py::test_year_between1990and1994_distance_morethan4000miles	0.12	
Passed *(show details)*	test_airline_stats.py::test_year_between1995and1999_distance_lessthan1000miles	0.12	
Passed *(show details)*	test_airline_stats.py::test_year_between1995and1999_distance_between1000and4000miles	0.14	

Result	Test	Duration	Links
Passed *(show details)*	test_airline_stats.py::test_year_between1995and1999_distance_morethan4000miles	0.09	
Passed *(show details)*	test_airline_stats.py::test_year_between2000and2004_distance_lessthan1000miles	0.13	
Passed *(show details)*	test_airline_stats.py::test_year_between2000and2004_distance_between1000and4000miles	0.11	
Passed *(show details)*	test_airline_stats.py::test_year_between2000and2004_distance_morethan4000miles	0.12	
Passed *(show details)*	test_airline_stats.py::test_year_between2005and2009_distance_lessthan1000miles	0.09	
Passed *(show details)*	test_airline_stats.py::test_year_between2005and2009_distance_between1000and4000miles	0.11	
Passed *(show details)*	test_airline_stats.py::test_year_between2005and2009_distance_morethan4000miles	0.10	
Passed *(show details)*	test_airline_stats.py::test_year_between2010and2014_distance_lessthan1000miles	0.11	
Passed *(show details)*	test_airline_stats.py::test_year_between2010and2014_distance_between1000and4000miles	0.09	
Passed *(show details)*	test_airline_stats.py::test_year_between2010and2014_distance_morethan4000miles	0.09	
Passed *(show details)*	test_airline_stats.py::test_year_between2015and2019_distance_lessthan1000miles	4.94	
Passed *(show details)*	test_airline_stats.py::test_year_between2015and2019_distance_between1000and4000miles	4.07	
Passed *(show details)*	test_airline_stats.py::test_year_between2015and2019_distance_morethan4000miles	3.84	
Passed *(show details)*	test_airline_stats.py::test_year_between2020and2021_distance_lessthan1000miles	2.27	
Passed *(show details)*	test_airline_stats.py::test_year_between2020and2021_distance_between1000and4000miles	1.17	
Passed *(show details)*	test_airline_stats.py::test_year_between2020and2021_distance_morethan4000miles	1.19	
Passed *(show details)*	test_delays_comparison.py::test_year_between2015and2019_distance_lessthan1000miles	23.52	
Passed *(show details)*	test_delays_comparison.py::test_year_between2015and2019_distance_between1000and4000miles	15.47	
Passed *(show details)*	test_delays_comparison.py::test_year_between2020and2021_distance_lessthan1000miles	2.75	
Failed *(hide details)*	test_airline_delays.py::test_year_between1990and1994_distance_lessthan1000miles	1.13	

```
self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
```

Result	Test	Duration	Links

```
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
>           return complexjson.loads(self.text, **kwargs)

venv/lib/python3.9/site-packages/requests/models.py:971:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
encoding = None, cls = None, object_hook = None, parse_float = None, parse_int = None, parse_constant = None, object_pairs_hook = None
use_decimal = False, kw = {}

    def loads(s, encoding=None, cls=None, object_hook=None, parse_float=None,
            parse_int=None, parse_constant=None, object_pairs_hook=None,
            use_decimal=False, **kw):
        """Deserialize ``s`` (a ``str`` or ``unicode`` instance containing a JSON
        document) to a Python object.

        *encoding* determines the encoding used to interpret any
        :class:`bytes` objects decoded by this instance (``'utf-8'`` by
        default). It has no effect when decoding :class:`unicode` objects.

        *object_hook*, if specified, will be called with the result of every
        JSON object decoded and its return value will be used in place of the
        given :class:`dict`.  This can be used to provide custom
        deserializations (e.g. to support JSON-RPC class hinting).

        *object_pairs_hook* is an optional function that will be called with
        the result of any object literal decode with an ordered list of pairs.
        The return value of *object_pairs_hook* will be used instead of the
        :class:`dict`.  This feature can be used to implement custom decoders
        that rely on the order that the key and value pairs are decoded (for
        example, :func:`collections.OrderedDict` will remember the order of
        insertion). If *object_hook* is also defined, the *object_pairs_hook*
        takes priority.
```

Result	Test	Duration	Links

```
        *parse_float*, if specified, will be called with the string of every
        JSON float to be decoded.  By default, this is equivalent to
        ``float(num_str)``. This can be used to use another datatype or parser
        for JSON floats (e.g. :class:`decimal.Decimal`).

        *parse_int*, if specified, will be called with the string of every
        JSON int to be decoded.  By default, this is equivalent to
        ``int(num_str)``.  This can be used to use another datatype or parser
        for JSON integers (e.g. :class:`float`).

        *parse_constant*, if specified, will be called with one of the
        following strings: ``'-Infinity'``, ``'Infinity'``, ``'NaN'``.  This
        can be used to raise an exception if invalid JSON numbers are
        encountered.

        If *use_decimal* is true (default: ``False``) then it implies
        parse_float=decimal.Decimal for parity with ``dump``.

        To use a custom ``JSONDecoder`` subclass, specify it with the ``cls``
        kwarg. NOTE: You should use *object_hook* or *object_pairs_hook* instead
        of subclassing whenever possible.

        """
        if (cls is None and encoding is None and object_hook is None and
                parse_int is None and parse_float is None and
                parse_constant is None and object_pairs_hook is None
                and not use_decimal and not kw):
>           return _default_decoder.decode(s)

venv/lib/python3.9/site-packages/simplejson/__init__.py:525:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
_w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def decode(self, s, _w=WHITESPACE.match, _PY3=PY3):
        """Return the Python representation of ``s`` (a ``str`` or ``unicode``
        instance containing a JSON document)

        """
        if _PY3 and isinstance(s, bytes):
            s = str(s, self.encoding)
>       obj, end = self.raw_decode(s)

venv/lib/python3.9/site-packages/simplejson/decoder.py:370:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

Result	Test	Duration	Links

```
self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
idx = 0, _w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def raw_decode(self, s, idx=0, _w=WHITESPACE.match, _PY3=PY3):
        """Decode a JSON document from ``s`` (a ``str`` or ``unicode``
        beginning with a JSON document) and return a 2-tuple of the Python
        representation and the index in ``s`` where the document ended.
        Optionally, ``idx`` can be used to specify an offset in ``s`` where
        the JSON document begins.

        This can be used to decode a JSON document from a string that may
        have extraneous data at the end.

        """
        if idx < 0:
            # Ensure that raw_decode bails on negative indexes, the regex
            # would otherwise mask this behavior. #98
            raise JSONDecodeError('Expecting value', s, idx)
        if _PY3 and not isinstance(s, str):
            raise TypeError("Input string must be text, not bytes")
        # strip UTF-8 bom
        if len(s) > idx:
            ord0 = ord(s[idx])
            if ord0 == 0xfeff:
                idx += 1
            elif ord0 == 0xef and s[idx:idx + 3] == '\xef\xbb\xbf':
                idx += 3
>       return self.scan_once(s, idx=_w(s, idx).end())
E       simplejson.errors.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/simplejson/decoder.py:400: JSONDecodeError

During handling of the above exception, another exception occurred:

    def test_year_between1990and1994_distance_lessthan1000miles():

        params = {'o': "SAT",
                  'dst': "PHX",
                  'a': "VX",
                  'yf' : 1990,
                  'yt' : 1995}

        response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                params=params)

        expected_output = {}
```

Result	Test	Duration	Links

```
>       assert response.json() == expected_output

test_airline_delays.py:21:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
            return complexjson.loads(self.text, **kwargs)
        except JSONDecodeError as e:
            # Catch JSON-related errors and raise as requests.JSONDecodeError
            # This aliases json.JSONDecodeError and simplejson.JSONDecodeError
>           raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)
E           requests.exceptions.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/requests/models.py:975: JSONDecodeError
```

Failed *(hide details)*	test_airline_delays.py::test_year_between1990and1994_distance_between1000and4000miles	0.21	

```
self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
```

Result	Test	Duration	Links

```
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
>           return complexjson.loads(self.text, **kwargs)

venv/lib/python3.9/site-packages/requests/models.py:971:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
encoding = None, cls = None, object_hook = None, parse_float = None, parse_int = None, parse_constant = None, object_pairs_hook = None
use_decimal = False, kw = {}

    def loads(s, encoding=None, cls=None, object_hook=None, parse_float=None,
            parse_int=None, parse_constant=None, object_pairs_hook=None,
            use_decimal=False, **kw):
        """Deserialize ``s`` (a ``str`` or ``unicode`` instance containing a JSON
        document) to a Python object.

        *encoding* determines the encoding used to interpret any
        :class:`bytes` objects decoded by this instance (``'utf-8'`` by
        default). It has no effect when decoding :class:`unicode` objects.

        *object_hook*, if specified, will be called with the result of every
        JSON object decoded and its return value will be used in place of the
        given :class:`dict`.  This can be used to provide custom
```

Result	Test	Duration	Links

```
        deserializations (e.g. to support JSON-RPC class hinting).

        *object_pairs_hook* is an optional function that will be called with
        the result of any object literal decode with an ordered list of pairs.
        The return value of *object_pairs_hook* will be used instead of the
        :class:`dict`.  This feature can be used to implement custom decoders
        that rely on the order that the key and value pairs are decoded (for
        example, :func:`collections.OrderedDict` will remember the order of
        insertion). If *object_hook* is also defined, the *object_pairs_hook*
        takes priority.

        *parse_float*, if specified, will be called with the string of every
        JSON float to be decoded.  By default, this is equivalent to
        ``float(num_str)``. This can be used to use another datatype or parser
        for JSON floats (e.g. :class:`decimal.Decimal`).

        *parse_int*, if specified, will be called with the string of every
        JSON int to be decoded.  By default, this is equivalent to
        ``int(num_str)``.  This can be used to use another datatype or parser
        for JSON integers (e.g. :class:`float`).

        *parse_constant*, if specified, will be called with one of the
        following strings: ``'-Infinity'``, ``'Infinity'``, ``'NaN'``.  This
        can be used to raise an exception if invalid JSON numbers are
        encountered.

        If *use_decimal* is true (default: ``False``) then it implies
        parse_float=decimal.Decimal for parity with ``dump``.

        To use a custom ``JSONDecoder`` subclass, specify it with the ``cls``
        kwarg. NOTE: You should use *object_hook* or *object_pairs_hook* instead
        of subclassing whenever possible.

        """
        if (cls is None and encoding is None and object_hook is None and
                parse_int is None and parse_float is None and
                parse_constant is None and object_pairs_hook is None
                and not use_decimal and not kw):
>           return _default_decoder.decode(s)

venv/lib/python3.9/site-packages/simplejson/__init__.py:525:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
_w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def decode(self, s, _w=WHITESPACE.match, _PY3=PY3):
```

Result	Test	Duration	Links

```
        """Return the Python representation of ``s`` (a ``str`` or ``unicode``
        instance containing a JSON document)

        """
        if _PY3 and isinstance(s, bytes):
            s = str(s, self.encoding)
>       obj, end = self.raw_decode(s)

venv/lib/python3.9/site-packages/simplejson/decoder.py:370:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
idx = 0, _w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def raw_decode(self, s, idx=0, _w=WHITESPACE.match, _PY3=PY3):
        """Decode a JSON document from ``s`` (a ``str`` or ``unicode``
        beginning with a JSON document) and return a 2-tuple of the Python
        representation and the index in ``s`` where the document ended.
        Optionally, ``idx`` can be used to specify an offset in ``s`` where
        the JSON document begins.

        This can be used to decode a JSON document from a string that may
        have extraneous data at the end.

        """
        if idx < 0:
            # Ensure that raw_decode bails on negative indexes, the regex
            # would otherwise mask this behavior. #98
            raise JSONDecodeError('Expecting value', s, idx)
        if _PY3 and not isinstance(s, str):
            raise TypeError("Input string must be text, not bytes")
        # strip UTF-8 bom
        if len(s) > idx:
            ord0 = ord(s[idx])
            if ord0 == 0xfeff:
                idx += 1
            elif ord0 == 0xef and s[idx:idx + 3] == '\xef\xbb\xbf':
                idx += 3
>       return self.scan_once(s, idx=_w(s, idx).end())
E       simplejson.errors.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/simplejson/decoder.py:400: JSONDecodeError

During handling of the above exception, another exception occurred:

    def test_year_between1990and1994_distance_between1000and4000miles():
```

Result	Test	Duration	Links

```
        params = {'o': "SFO",
                  'dst': "EWR",
                  'a': "US",
                  'yf' : 1990,
                  'yt' : 1994}

        response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                params=params)

        expected_output = {}

>       assert response.json() == expected_output

test_airline_delays.py:38:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
            return complexjson.loads(self.text, **kwargs)
        except JSONDecodeError as e:
            # Catch JSON-related errors and raise as requests.JSONDecodeError
            # This aliases json.JSONDecodeError and simplejson.JSONDecodeError
```

Result	Test	Duration	Links

```
>           raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)
E           requests.exceptions.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/requests/models.py:975: JSONDecodeError
```

Failed *(hide details)*	test_airline_delays.py::test_year_between1990and1994_distance_morethan4000miles	0.17	

```
self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
>           return complexjson.loads(self.text, **kwargs)

venv/lib/python3.9/site-packages/requests/models.py:971:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
encoding = None, cls = None, object_hook = None, parse_float = None, parse_int = None, parse_constant = None, object_pairs_hook = None
use_decimal = False, kw = {}

    def loads(s, encoding=None, cls=None, object_hook=None, parse_float=None,
            parse_int=None, parse_constant=None, object_pairs_hook=None,
            use_decimal=False, **kw):
```

Result	Test	Duration	Links

```
        """Deserialize ``s`` (a ``str`` or ``unicode`` instance containing a JSON
        document) to a Python object.

        *encoding* determines the encoding used to interpret any
        :class:`bytes` objects decoded by this instance (``'utf-8'`` by
        default). It has no effect when decoding :class:`unicode` objects.

        *object_hook*, if specified, will be called with the result of every
        JSON object decoded and its return value will be used in place of the
        given :class:`dict`.  This can be used to provide custom
        deserializations (e.g. to support JSON-RPC class hinting).

        *object_pairs_hook* is an optional function that will be called with
        the result of any object literal decode with an ordered list of pairs.
        The return value of *object_pairs_hook* will be used instead of the
        :class:`dict`.  This feature can be used to implement custom decoders
        that rely on the order that the key and value pairs are decoded (for
        example, :func:`collections.OrderedDict` will remember the order of
        insertion). If *object_hook* is also defined, the *object_pairs_hook*
        takes priority.

        *parse_float*, if specified, will be called with the string of every
        JSON float to be decoded.  By default, this is equivalent to
        ``float(num_str)``. This can be used to use another datatype or parser
        for JSON floats (e.g. :class:`decimal.Decimal`).

        *parse_int*, if specified, will be called with the string of every
        JSON int to be decoded.  By default, this is equivalent to
        ``int(num_str)``.  This can be used to use another datatype or parser
        for JSON integers (e.g. :class:`float`).

        *parse_constant*, if specified, will be called with one of the
        following strings: ``'-Infinity'``, ``'Infinity'``, ``'NaN'``.  This
        can be used to raise an exception if invalid JSON numbers are
        encountered.

        If *use_decimal* is true (default: ``False``) then it implies
        parse_float=decimal.Decimal for parity with ``dump``.

        To use a custom ``JSONDecoder`` subclass, specify it with the ``cls``
        kwarg. NOTE: You should use *object_hook* or *object_pairs_hook* instead
        of subclassing whenever possible.

        """
        if (cls is None and encoding is None and object_hook is None and
                parse_int is None and parse_float is None and
                parse_constant is None and object_pairs_hook is None
                and not use_decimal and not kw):
>           return _default_decoder.decode(s)
```

Result	Test	Duration	Links

```
venv/lib/python3.9/site-packages/simplejson/__init__.py:525:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
_w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def decode(self, s, _w=WHITESPACE.match, _PY3=PY3):
        """Return the Python representation of ``s`` (a ``str`` or ``unicode``
        instance containing a JSON document)

        """
        if _PY3 and isinstance(s, bytes):
            s = str(s, self.encoding)
>       obj, end = self.raw_decode(s)

venv/lib/python3.9/site-packages/simplejson/decoder.py:370:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
idx = 0, _w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def raw_decode(self, s, idx=0, _w=WHITESPACE.match, _PY3=PY3):
        """Decode a JSON document from ``s`` (a ``str`` or ``unicode``
        beginning with a JSON document) and return a 2-tuple of the Python
        representation and the index in ``s`` where the document ended.
        Optionally, ``idx`` can be used to specify an offset in ``s`` where
        the JSON document begins.

        This can be used to decode a JSON document from a string that may
        have extraneous data at the end.

        """
        if idx < 0:
            # Ensure that raw_decode bails on negative indexes, the regex
            # would otherwise mask this behavior. #98
            raise JSONDecodeError('Expecting value', s, idx)
        if _PY3 and not isinstance(s, str):
            raise TypeError("Input string must be text, not bytes")
        # strip UTF-8 bom
        if len(s) > idx:
            ord0 = ord(s[idx])
            if ord0 == 0xfeff:
                idx += 1
            elif ord0 == 0xef and s[idx:idx + 3] == '\xef\xbb\xbf':
```

Result	Test	Duration	Links

```
                idx += 3
>       return self.scan_once(s, idx=_w(s, idx).end())
E       simplejson.errors.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/simplejson/decoder.py:400: JSONDecodeError

During handling of the above exception, another exception occurred:

    def test_year_between1990and1994_distance_morethan4000miles():

        params = {'o': "HNL",
                  'dst': "JFK",
                  'a': "DL",
                  'yf' : 1990,
                  'yt' : 1994}

        response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                params=params)

        expected_output = {}

>       assert response.json() == expected_output

test_airline_delays.py:54:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
```

Result	Test	Duration	Links

```
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
            return complexjson.loads(self.text, **kwargs)
        except JSONDecodeError as e:
            # Catch JSON-related errors and raise as requests.JSONDecodeError
            # This aliases json.JSONDecodeError and simplejson.JSONDecodeError
>           raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)
E           requests.exceptions.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/requests/models.py:975: JSONDecodeError
```

Failed *(hide details)*	test_airline_delays.py::test_year_between1995and1999_distance_lessthan1000miles	1.18	

```
self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
>           return complexjson.loads(self.text, **kwargs)

venv/lib/python3.9/site-packages/requests/models.py:971:
```

Result	Test	Duration	Links

```
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
encoding = None, cls = None, object_hook = None, parse_float = None, parse_int = None, parse_constant = None, object_pairs_hook = None
use_decimal = False, kw = {}

    def loads(s, encoding=None, cls=None, object_hook=None, parse_float=None,
            parse_int=None, parse_constant=None, object_pairs_hook=None,
            use_decimal=False, **kw):
        """Deserialize ``s`` (a ``str`` or ``unicode`` instance containing a JSON
        document) to a Python object.

        *encoding* determines the encoding used to interpret any
        :class:`bytes` objects decoded by this instance (``'utf-8'`` by
        default). It has no effect when decoding :class:`unicode` objects.

        *object_hook*, if specified, will be called with the result of every
        JSON object decoded and its return value will be used in place of the
        given :class:`dict`.  This can be used to provide custom
        deserializations (e.g. to support JSON-RPC class hinting).

        *object_pairs_hook* is an optional function that will be called with
        the result of any object literal decode with an ordered list of pairs.
        The return value of *object_pairs_hook* will be used instead of the
        :class:`dict`.  This feature can be used to implement custom decoders
        that rely on the order that the key and value pairs are decoded (for
        example, :func:`collections.OrderedDict` will remember the order of
        insertion). If *object_hook* is also defined, the *object_pairs_hook*
        takes priority.

        *parse_float*, if specified, will be called with the string of every
        JSON float to be decoded.  By default, this is equivalent to
        ``float(num_str)``. This can be used to use another datatype or parser
        for JSON floats (e.g. :class:`decimal.Decimal`).

        *parse_int*, if specified, will be called with the string of every
        JSON int to be decoded.  By default, this is equivalent to
        ``int(num_str)``.  This can be used to use another datatype or parser
        for JSON integers (e.g. :class:`float`).

        *parse_constant*, if specified, will be called with one of the
        following strings: ``'-Infinity'``, ``'Infinity'``, ``'NaN'``.  This
        can be used to raise an exception if invalid JSON numbers are
        encountered.

        If *use_decimal* is true (default: ``False``) then it implies
        parse_float=decimal.Decimal for parity with ``dump``.
```

Result	Test	Duration	Links

```
        To use a custom ``JSONDecoder`` subclass, specify it with the ``cls``
        kwarg. NOTE: You should use *object_hook* or *object_pairs_hook* instead
        of subclassing whenever possible.

        """
        if (cls is None and encoding is None and object_hook is None and
                parse_int is None and parse_float is None and
                parse_constant is None and object_pairs_hook is None
                and not use_decimal and not kw):
>           return _default_decoder.decode(s)

venv/lib/python3.9/site-packages/simplejson/__init__.py:525:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
_w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def decode(self, s, _w=WHITESPACE.match, _PY3=PY3):
        """Return the Python representation of ``s`` (a ``str`` or ``unicode``
        instance containing a JSON document)

        """
        if _PY3 and isinstance(s, bytes):
            s = str(s, self.encoding)
>       obj, end = self.raw_decode(s)

venv/lib/python3.9/site-packages/simplejson/decoder.py:370:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
idx = 0, _w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def raw_decode(self, s, idx=0, _w=WHITESPACE.match, _PY3=PY3):
        """Decode a JSON document from ``s`` (a ``str`` or ``unicode``
        beginning with a JSON document) and return a 2-tuple of the Python
        representation and the index in ``s`` where the document ended.
        Optionally, ``idx`` can be used to specify an offset in ``s`` where
        the JSON document begins.

        This can be used to decode a JSON document from a string that may
        have extraneous data at the end.

        """
        if idx < 0:
            # Ensure that raw_decode bails on negative indexes, the regex
```

Result	Test	Duration	Links

```
            # would otherwise mask this behavior. #98
            raise JSONDecodeError('Expecting value', s, idx)
        if _PY3 and not isinstance(s, str):
            raise TypeError("Input string must be text, not bytes")
        # strip UTF-8 bom
        if len(s) > idx:
            ord0 = ord(s[idx])
            if ord0 == 0xfeff:
                idx += 1
            elif ord0 == 0xef and s[idx:idx + 3] == '\xef\xbb\xbf':
                idx += 3
>       return self.scan_once(s, idx=_w(s, idx).end())
E       simplejson.errors.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/simplejson/decoder.py:400: JSONDecodeError

During handling of the above exception, another exception occurred:

    def test_year_between1995and1999_distance_lessthan1000miles():

        params = {'o': "LGA",
                  'dst': "ORD",
                  'a': "AX",
                  'yf' : 1995,
                  'yt' : 1999}

        response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                params=params)

        expected_output = {}

>       assert response.json() == expected_output

test_airline_delays.py:70:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
```

Result	Test	Duration	Links

```
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
            return complexjson.loads(self.text, **kwargs)
        except JSONDecodeError as e:
            # Catch JSON-related errors and raise as requests.JSONDecodeError
            # This aliases json.JSONDecodeError and simplejson.JSONDecodeError
>           raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)
E           requests.exceptions.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/requests/models.py:975: JSONDecodeError
```

Failed *(hide details)*	test_airline_delays.py::test_year_between1995and1999_distance_between1000and4000miles	0.16	

```
self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
```

Result	Test	Duration	Links

```
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
>           return complexjson.loads(self.text, **kwargs)

venv/lib/python3.9/site-packages/requests/models.py:971:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
encoding = None, cls = None, object_hook = None, parse_float = None, parse_int = None, parse_constant = None, object_pairs_hook = None
use_decimal = False, kw = {}

    def loads(s, encoding=None, cls=None, object_hook=None, parse_float=None,
            parse_int=None, parse_constant=None, object_pairs_hook=None,
            use_decimal=False, **kw):
        """Deserialize ``s`` (a ``str`` or ``unicode`` instance containing a JSON
        document) to a Python object.

        *encoding* determines the encoding used to interpret any
        :class:`bytes` objects decoded by this instance (``'utf-8'`` by
        default). It has no effect when decoding :class:`unicode` objects.

        *object_hook*, if specified, will be called with the result of every
        JSON object decoded and its return value will be used in place of the
        given :class:`dict`.  This can be used to provide custom
        deserializations (e.g. to support JSON-RPC class hinting).

        *object_pairs_hook* is an optional function that will be called with
        the result of any object literal decode with an ordered list of pairs.
        The return value of *object_pairs_hook* will be used instead of the
        :class:`dict`.  This feature can be used to implement custom decoders
        that rely on the order that the key and value pairs are decoded (for
        example, :func:`collections.OrderedDict` will remember the order of
        insertion). If *object_hook* is also defined, the *object_pairs_hook*
        takes priority.

        *parse_float*, if specified, will be called with the string of every
        JSON float to be decoded.  By default, this is equivalent to
        ``float(num_str)``. This can be used to use another datatype or parser
        for JSON floats (e.g. :class:`decimal.Decimal`).

        *parse_int*, if specified, will be called with the string of every
        JSON int to be decoded.  By default, this is equivalent to
        ``int(num_str)``.  This can be used to use another datatype or parser
```

Result	Test	Duration	Links

```
        for JSON integers (e.g. :class:`float`).

        *parse_constant*, if specified, will be called with one of the
        following strings: ``'-Infinity'``, ``'Infinity'``, ``'NaN'``.  This
        can be used to raise an exception if invalid JSON numbers are
        encountered.

        If *use_decimal* is true (default: ``False``) then it implies
        parse_float=decimal.Decimal for parity with ``dump``.

        To use a custom ``JSONDecoder`` subclass, specify it with the ``cls``
        kwarg. NOTE: You should use *object_hook* or *object_pairs_hook* instead
        of subclassing whenever possible.

        """
        if (cls is None and encoding is None and object_hook is None and
                parse_int is None and parse_float is None and
                parse_constant is None and object_pairs_hook is None
                and not use_decimal and not kw):
>           return _default_decoder.decode(s)

venv/lib/python3.9/site-packages/simplejson/__init__.py:525:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
_w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def decode(self, s, _w=WHITESPACE.match, _PY3=PY3):
        """Return the Python representation of ``s`` (a ``str`` or ``unicode``
        instance containing a JSON document)

        """
        if _PY3 and isinstance(s, bytes):
            s = str(s, self.encoding)
>       obj, end = self.raw_decode(s)

venv/lib/python3.9/site-packages/simplejson/decoder.py:370:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
idx = 0, _w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def raw_decode(self, s, idx=0, _w=WHITESPACE.match, _PY3=PY3):
        """Decode a JSON document from ``s`` (a ``str`` or ``unicode``
        beginning with a JSON document) and return a 2-tuple of the Python
```

Result | Test | Duration | Links

```
        representation and the index in ``s`` where the document ended.
        Optionally, ``idx`` can be used to specify an offset in ``s`` where
        the JSON document begins.

        This can be used to decode a JSON document from a string that may
        have extraneous data at the end.

        """
        if idx < 0:
            # Ensure that raw_decode bails on negative indexes, the regex
            # would otherwise mask this behavior. #98
            raise JSONDecodeError('Expecting value', s, idx)
        if _PY3 and not isinstance(s, str):
            raise TypeError("Input string must be text, not bytes")
        # strip UTF-8 bom
        if len(s) > idx:
            ord0 = ord(s[idx])
            if ord0 == 0xfeff:
                idx += 1
            elif ord0 == 0xef and s[idx:idx + 3] == '\xef\xbb\xbf':
                idx += 3
>       return self.scan_once(s, idx=_w(s, idx).end())
E       simplejson.errors.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/simplejson/decoder.py:400: JSONDecodeError

During handling of the above exception, another exception occurred:

    def test_year_between1995and1999_distance_between1000and4000miles():

        params = {'o': "IAH",
                  'dst': "SEA",
                  'a': "NK",
                  'yf' : 1995,
                  'yt' : 1999}

        response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                params=params)

        expected_output = {}

>       assert response.json() == expected_output

test_airline_delays.py:86:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
```

Result	Test	Duration	Links

```
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
            return complexjson.loads(self.text, **kwargs)
        except JSONDecodeError as e:
            # Catch JSON-related errors and raise as requests.JSONDecodeError
            # This aliases json.JSONDecodeError and simplejson.JSONDecodeError
>           raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)
E           requests.exceptions.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/requests/models.py:975: JSONDecodeError
```

Failed *(hide details)*	test_airline_delays.py::test_year_between1995and1999_distance_morethan4000miles	0.15	

```
self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
```

Result	Test	Duration	Links

```
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
>           return complexjson.loads(self.text, **kwargs)

venv/lib/python3.9/site-packages/requests/models.py:971:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
encoding = None, cls = None, object_hook = None, parse_float = None, parse_int = None, parse_constant = None, object_pairs_hook = None
use_decimal = False, kw = {}

    def loads(s, encoding=None, cls=None, object_hook=None, parse_float=None,
            parse_int=None, parse_constant=None, object_pairs_hook=None,
            use_decimal=False, **kw):
        """Deserialize ``s`` (a ``str`` or ``unicode`` instance containing a JSON
        document) to a Python object.

        *encoding* determines the encoding used to interpret any
        :class:`bytes` objects decoded by this instance (``'utf-8'`` by
        default). It has no effect when decoding :class:`unicode` objects.

        *object_hook*, if specified, will be called with the result of every
        JSON object decoded and its return value will be used in place of the
        given :class:`dict`.  This can be used to provide custom
        deserializations (e.g. to support JSON-RPC class hinting).

        *object_pairs_hook* is an optional function that will be called with
        the result of any object literal decode with an ordered list of pairs.
        The return value of *object_pairs_hook* will be used instead of the
        :class:`dict`.  This feature can be used to implement custom decoders
        that rely on the order that the key and value pairs are decoded (for
        example, :func:`collections.OrderedDict` will remember the order of
        insertion). If *object_hook* is also defined, the *object_pairs_hook*
```

Result	Test	Duration	Links

```
        takes priority.

        *parse_float*, if specified, will be called with the string of every
        JSON float to be decoded.  By default, this is equivalent to
        ``float(num_str)``. This can be used to use another datatype or parser
        for JSON floats (e.g. :class:`decimal.Decimal`).

        *parse_int*, if specified, will be called with the string of every
        JSON int to be decoded.  By default, this is equivalent to
        ``int(num_str)``.  This can be used to use another datatype or parser
        for JSON integers (e.g. :class:`float`).

        *parse_constant*, if specified, will be called with one of the
        following strings: ``'-Infinity'``, ``'Infinity'``, ``'NaN'``.  This
        can be used to raise an exception if invalid JSON numbers are
        encountered.

        If *use_decimal* is true (default: ``False``) then it implies
        parse_float=decimal.Decimal for parity with ``dump``.

        To use a custom ``JSONDecoder`` subclass, specify it with the ``cls``
        kwarg. NOTE: You should use *object_hook* or *object_pairs_hook* instead
        of subclassing whenever possible.

        """
        if (cls is None and encoding is None and object_hook is None and
                parse_int is None and parse_float is None and
                parse_constant is None and object_pairs_hook is None
                and not use_decimal and not kw):
>           return _default_decoder.decode(s)

venv/lib/python3.9/site-packages/simplejson/__init__.py:525:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
_w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def decode(self, s, _w=WHITESPACE.match, _PY3=PY3):
        """Return the Python representation of ``s`` (a ``str`` or ``unicode``
        instance containing a JSON document)

        """
        if _PY3 and isinstance(s, bytes):
            s = str(s, self.encoding)
>       obj, end = self.raw_decode(s)

venv/lib/python3.9/site-packages/simplejson/decoder.py:370:
```

Result	Test	Duration	Links

```
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
idx = 0, _w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def raw_decode(self, s, idx=0, _w=WHITESPACE.match, _PY3=PY3):
        """Decode a JSON document from ``s`` (a ``str`` or ``unicode``
        beginning with a JSON document) and return a 2-tuple of the Python
        representation and the index in ``s`` where the document ended.
        Optionally, ``idx`` can be used to specify an offset in ``s`` where
        the JSON document begins.

        This can be used to decode a JSON document from a string that may
        have extraneous data at the end.

        """
        if idx < 0:
            # Ensure that raw_decode bails on negative indexes, the regex
            # would otherwise mask this behavior. #98
            raise JSONDecodeError('Expecting value', s, idx)
        if _PY3 and not isinstance(s, str):
            raise TypeError("Input string must be text, not bytes")
        # strip UTF-8 bom
        if len(s) > idx:
            ord0 = ord(s[idx])
            if ord0 == 0xfeff:
                idx += 1
            elif ord0 == 0xef and s[idx:idx + 3] == '\xef\xbb\xbf':
                idx += 3
>       return self.scan_once(s, idx=_w(s, idx).end())
E       simplejson.errors.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/simplejson/decoder.py:400: JSONDecodeError

During handling of the above exception, another exception occurred:

    def test_year_between1995and1999_distance_morethan4000miles():

        params = {'o': "HNL",
                  'dst': "ATL",
                  'a': "EM",
                  'yf' : 1995,
                  'yt' : 1999}

        response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                params=params)
```

Result	Test	Duration	Links

```
        expected_output = {}

>       assert response.json() == expected_output

test_airline_delays.py:102:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
            return complexjson.loads(self.text, **kwargs)
        except JSONDecodeError as e:
            # Catch JSON-related errors and raise as requests.JSONDecodeError
            # This aliases json.JSONDecodeError and simplejson.JSONDecodeError
>           raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)
E           requests.exceptions.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/requests/models.py:975: JSONDecodeError
```

Failed *(hide details)*	test_airline_delays.py::test_year_between2000and2004_distance_lessthan1000miles	0.15	

```
self = <Response [500]>, kwargs = {}
```

Result	Test	Duration	Links

```
    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
>           return complexjson.loads(self.text, **kwargs)

venv/lib/python3.9/site-packages/requests/models.py:971:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
encoding = None, cls = None, object_hook = None, parse_float = None, parse_int = None, parse_constant = None, object_pairs_hook = None
use_decimal = False, kw = {}

    def loads(s, encoding=None, cls=None, object_hook=None, parse_float=None,
            parse_int=None, parse_constant=None, object_pairs_hook=None,
            use_decimal=False, **kw):
        """Deserialize ``s`` (a ``str`` or ``unicode`` instance containing a JSON
        document) to a Python object.

        *encoding* determines the encoding used to interpret any
        :class:`bytes` objects decoded by this instance (``'utf-8'`` by
        default). It has no effect when decoding :class:`unicode` objects.

        *object_hook*, if specified, will be called with the result of every
        JSON object decoded and its return value will be used in place of the
```

Result	Test	Duration	Links

```
        given :class:`dict`.  This can be used to provide custom
        deserializations (e.g. to support JSON-RPC class hinting).

        *object_pairs_hook* is an optional function that will be called with
        the result of any object literal decode with an ordered list of pairs.
        The return value of *object_pairs_hook* will be used instead of the
        :class:`dict`.  This feature can be used to implement custom decoders
        that rely on the order that the key and value pairs are decoded (for
        example, :func:`collections.OrderedDict` will remember the order of
        insertion). If *object_hook* is also defined, the *object_pairs_hook*
        takes priority.

        *parse_float*, if specified, will be called with the string of every
        JSON float to be decoded.  By default, this is equivalent to
        ``float(num_str)``. This can be used to use another datatype or parser
        for JSON floats (e.g. :class:`decimal.Decimal`).

        *parse_int*, if specified, will be called with the string of every
        JSON int to be decoded.  By default, this is equivalent to
        ``int(num_str)``.  This can be used to use another datatype or parser
        for JSON integers (e.g. :class:`float`).

        *parse_constant*, if specified, will be called with one of the
        following strings: ``'-Infinity'``, ``'Infinity'``, ``'NaN'``.  This
        can be used to raise an exception if invalid JSON numbers are
        encountered.

        If *use_decimal* is true (default: ``False``) then it implies
        parse_float=decimal.Decimal for parity with ``dump``.

        To use a custom ``JSONDecoder`` subclass, specify it with the ``cls``
        kwarg. NOTE: You should use *object_hook* or *object_pairs_hook* instead
        of subclassing whenever possible.

        """
        if (cls is None and encoding is None and object_hook is None and
                parse_int is None and parse_float is None and
                parse_constant is None and object_pairs_hook is None
                and not use_decimal and not kw):
>           return _default_decoder.decode(s)

venv/lib/python3.9/site-packages/simplejson/__init__.py:525:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
_w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True
```

Result	Test	Duration	Links

```
    def decode(self, s, _w=WHITESPACE.match, _PY3=PY3):
        """Return the Python representation of ``s`` (a ``str`` or ``unicode``
        instance containing a JSON document)

        """
        if _PY3 and isinstance(s, bytes):
            s = str(s, self.encoding)
>       obj, end = self.raw_decode(s)

venv/lib/python3.9/site-packages/simplejson/decoder.py:370:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
idx = 0, _w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def raw_decode(self, s, idx=0, _w=WHITESPACE.match, _PY3=PY3):
        """Decode a JSON document from ``s`` (a ``str`` or ``unicode``
        beginning with a JSON document) and return a 2-tuple of the Python
        representation and the index in ``s`` where the document ended.
        Optionally, ``idx`` can be used to specify an offset in ``s`` where
        the JSON document begins.

        This can be used to decode a JSON document from a string that may
        have extraneous data at the end.

        """
        if idx < 0:
            # Ensure that raw_decode bails on negative indexes, the regex
            # would otherwise mask this behavior. #98
            raise JSONDecodeError('Expecting value', s, idx)
        if _PY3 and not isinstance(s, str):
            raise TypeError("Input string must be text, not bytes")
        # strip UTF-8 bom
        if len(s) > idx:
            ord0 = ord(s[idx])
            if ord0 == 0xfeff:
                idx += 1
            elif ord0 == 0xef and s[idx:idx + 3] == '\xef\xbb\xbf':
                idx += 3
>       return self.scan_once(s, idx=_w(s, idx).end())
E       simplejson.errors.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/simplejson/decoder.py:400: JSONDecodeError

During handling of the above exception, another exception occurred:

    def test_year_between2000and2004_distance_lessthan1000miles():
```

Result	Test	Duration	Links

```
        params = {'o': "DFW",
                  'dst': "MKE",
                  'a': "G7",
                  'yf' : 2000,
                  'yt' : 2004}

        response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                params=params)

        expected_output = {}

>       assert response.json() == expected_output

test_airline_delays.py:118:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
            return complexjson.loads(self.text, **kwargs)
        except JSONDecodeError as e:
            # Catch JSON-related errors and raise as requests.JSONDecodeError
```

Result	Test	Duration	Links

```
            # This aliases json.JSONDecodeError and simplejson.JSONDecodeError
>           raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)
E           requests.exceptions.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/requests/models.py:975: JSONDecodeError
```

Failed *(hide details)*	test_airline_delays.py::test_year_between2000and2004_distance_between1000and4000miles	0.21	

```
self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
>           return complexjson.loads(self.text, **kwargs)

venv/lib/python3.9/site-packages/requests/models.py:971:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
encoding = None, cls = None, object_hook = None, parse_float = None, parse_int = None, parse_constant = None, object_pairs_hook = None
use_decimal = False, kw = {}

    def loads(s, encoding=None, cls=None, object_hook=None, parse_float=None,
            parse_int=None, parse_constant=None, object_pairs_hook=None,
```

Result	Test	Duration	Links

```
    use_decimal=False, **kw):
"""Deserialize ``s`` (a ``str`` or ``unicode`` instance containing a JSON
document) to a Python object.

*encoding* determines the encoding used to interpret any
:class:`bytes` objects decoded by this instance (``'utf-8'`` by
default). It has no effect when decoding :class:`unicode` objects.

*object_hook*, if specified, will be called with the result of every
JSON object decoded and its return value will be used in place of the
given :class:`dict`.  This can be used to provide custom
deserializations (e.g. to support JSON-RPC class hinting).

*object_pairs_hook* is an optional function that will be called with
the result of any object literal decode with an ordered list of pairs.
The return value of *object_pairs_hook* will be used instead of the
:class:`dict`.  This feature can be used to implement custom decoders
that rely on the order that the key and value pairs are decoded (for
example, :func:`collections.OrderedDict` will remember the order of
insertion). If *object_hook* is also defined, the *object_pairs_hook*
takes priority.

*parse_float*, if specified, will be called with the string of every
JSON float to be decoded.  By default, this is equivalent to
``float(num_str)``. This can be used to use another datatype or parser
for JSON floats (e.g. :class:`decimal.Decimal`).

*parse_int*, if specified, will be called with the string of every
JSON int to be decoded.  By default, this is equivalent to
``int(num_str)``.  This can be used to use another datatype or parser
for JSON integers (e.g. :class:`float`).

*parse_constant*, if specified, will be called with one of the
following strings: ``'-Infinity'``, ``'Infinity'``, ``'NaN'``.  This
can be used to raise an exception if invalid JSON numbers are
encountered.

If *use_decimal* is true (default: ``False``) then it implies
parse_float=decimal.Decimal for parity with ``dump``.

To use a custom ``JSONDecoder`` subclass, specify it with the ``cls``
kwarg. NOTE: You should use *object_hook* or *object_pairs_hook* instead
of subclassing whenever possible.

"""
if (cls is None and encoding is None and object_hook is None and
        parse_int is None and parse_float is None and
        parse_constant is None and object_pairs_hook is None
        and not use_decimal and not kw):
```

Result	Test	Duration	Links

```
>           return _default_decoder.decode(s)

venv/lib/python3.9/site-packages/simplejson/__init__.py:525:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
_w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def decode(self, s, _w=WHITESPACE.match, _PY3=PY3):
        """Return the Python representation of ``s`` (a ``str`` or ``unicode``
        instance containing a JSON document)

        """
        if _PY3 and isinstance(s, bytes):
            s = str(s, self.encoding)
>       obj, end = self.raw_decode(s)

venv/lib/python3.9/site-packages/simplejson/decoder.py:370:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
idx = 0, _w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def raw_decode(self, s, idx=0, _w=WHITESPACE.match, _PY3=PY3):
        """Decode a JSON document from ``s`` (a ``str`` or ``unicode``
        beginning with a JSON document) and return a 2-tuple of the Python
        representation and the index in ``s`` where the document ended.
        Optionally, ``idx`` can be used to specify an offset in ``s`` where
        the JSON document begins.

        This can be used to decode a JSON document from a string that may
        have extraneous data at the end.

        """
        if idx < 0:
            # Ensure that raw_decode bails on negative indexes, the regex
            # would otherwise mask this behavior. #98
            raise JSONDecodeError('Expecting value', s, idx)
        if _PY3 and not isinstance(s, str):
            raise TypeError("Input string must be text, not bytes")
        # strip UTF-8 bom
        if len(s) > idx:
            ord0 = ord(s[idx])
            if ord0 == 0xfeff:
                idx += 1
```

Result	Test	Duration	Links

```
            elif ord0 == 0xef and s[idx:idx + 3] == '\xef\xbb\xbf':
                idx += 3
>       return self.scan_once(s, idx=_w(s, idx).end())
E       simplejson.errors.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/simplejson/decoder.py:400: JSONDecodeError

During handling of the above exception, another exception occurred:

    def test_year_between2000and2004_distance_between1000and4000miles():

        params = {'o': "HNL",
                  'dst': "LAX",
                  'a': "YX",
                  'yf' : 2000,
                  'yt' : 2004}

        response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                params=params)

        expected_output = {}

>       assert response.json() == expected_output

test_airline_delays.py:134:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
```

Result	Test	Duration	Links

```
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
            return complexjson.loads(self.text, **kwargs)
        except JSONDecodeError as e:
            # Catch JSON-related errors and raise as requests.JSONDecodeError
            # This aliases json.JSONDecodeError and simplejson.JSONDecodeError
>           raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)
E           requests.exceptions.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/requests/models.py:975: JSONDecodeError
```

Failed *(hide details)*	test_airline_delays.py::test_year_between2000and2004_distance_morethan4000miles	0.15	

```
self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
>           return complexjson.loads(self.text, **kwargs)
```

Result	Test	Duration	Links

```
venv/lib/python3.9/site-packages/requests/models.py:971:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
encoding = None, cls = None, object_hook = None, parse_float = None, parse_int = None, parse_constant = None, object_pairs_hook = None
use_decimal = False, kw = {}

    def loads(s, encoding=None, cls=None, object_hook=None, parse_float=None,
            parse_int=None, parse_constant=None, object_pairs_hook=None,
            use_decimal=False, **kw):
        """Deserialize ``s`` (a ``str`` or ``unicode`` instance containing a JSON
        document) to a Python object.

        *encoding* determines the encoding used to interpret any
        :class:`bytes` objects decoded by this instance (``'utf-8'`` by
        default). It has no effect when decoding :class:`unicode` objects.

        *object_hook*, if specified, will be called with the result of every
        JSON object decoded and its return value will be used in place of the
        given :class:`dict`.  This can be used to provide custom
        deserializations (e.g. to support JSON-RPC class hinting).

        *object_pairs_hook* is an optional function that will be called with
        the result of any object literal decode with an ordered list of pairs.
        The return value of *object_pairs_hook* will be used instead of the
        :class:`dict`.  This feature can be used to implement custom decoders
        that rely on the order that the key and value pairs are decoded (for
        example, :func:`collections.OrderedDict` will remember the order of
        insertion). If *object_hook* is also defined, the *object_pairs_hook*
        takes priority.

        *parse_float*, if specified, will be called with the string of every
        JSON float to be decoded.  By default, this is equivalent to
        ``float(num_str)``. This can be used to use another datatype or parser
        for JSON floats (e.g. :class:`decimal.Decimal`).

        *parse_int*, if specified, will be called with the string of every
        JSON int to be decoded.  By default, this is equivalent to
        ``int(num_str)``.  This can be used to use another datatype or parser
        for JSON integers (e.g. :class:`float`).

        *parse_constant*, if specified, will be called with one of the
        following strings: ``'-Infinity'``, ``'Infinity'``, ``'NaN'``.  This
        can be used to raise an exception if invalid JSON numbers are
        encountered.

        If *use_decimal* is true (default: ``False``) then it implies
        parse_float=decimal.Decimal for parity with ``dump``.
```

Result	Test	Duration	Links

```
        To use a custom ``JSONDecoder`` subclass, specify it with the ``cls``
        kwarg. NOTE: You should use *object_hook* or *object_pairs_hook* instead
        of subclassing whenever possible.

        """
        if (cls is None and encoding is None and object_hook is None and
                parse_int is None and parse_float is None and
                parse_constant is None and object_pairs_hook is None
                and not use_decimal and not kw):
>           return _default_decoder.decode(s)

venv/lib/python3.9/site-packages/simplejson/__init__.py:525:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
_w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def decode(self, s, _w=WHITESPACE.match, _PY3=PY3):
        """Return the Python representation of ``s`` (a ``str`` or ``unicode``
        instance containing a JSON document)

        """
        if _PY3 and isinstance(s, bytes):
            s = str(s, self.encoding)
>       obj, end = self.raw_decode(s)

venv/lib/python3.9/site-packages/simplejson/decoder.py:370:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
idx = 0, _w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def raw_decode(self, s, idx=0, _w=WHITESPACE.match, _PY3=PY3):
        """Decode a JSON document from ``s`` (a ``str`` or ``unicode``
        beginning with a JSON document) and return a 2-tuple of the Python
        representation and the index in ``s`` where the document ended.
        Optionally, ``idx`` can be used to specify an offset in ``s`` where
        the JSON document begins.

        This can be used to decode a JSON document from a string that may
        have extraneous data at the end.

        """
        if idx < 0:
```

Result	Test	Duration	Links

```
            # Ensure that raw_decode bails on negative indexes, the regex
            # would otherwise mask this behavior. #98
            raise JSONDecodeError('Expecting value', s, idx)
        if _PY3 and not isinstance(s, str):
            raise TypeError("Input string must be text, not bytes")
        # strip UTF-8 bom
        if len(s) > idx:
            ord0 = ord(s[idx])
            if ord0 == 0xfeff:
                idx += 1
            elif ord0 == 0xef and s[idx:idx + 3] == '\xef\xbb\xbf':
                idx += 3
>       return self.scan_once(s, idx=_w(s, idx).end())
E       simplejson.errors.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/simplejson/decoder.py:400: JSONDecodeError

During handling of the above exception, another exception occurred:

    def test_year_between2000and2004_distance_morethan4000miles():

        params = {'o': "IAH",
                  'dst': "HNL",
                  'a': "OH",
                  'yf' : 2000,
                  'yt' : 2004}

        response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                params=params)

        expected_output = {}

>       assert response.json() == expected_output

test_airline_delays.py:150:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
```

Result	Test	Duration	Links

```
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
            return complexjson.loads(self.text, **kwargs)
        except JSONDecodeError as e:
            # Catch JSON-related errors and raise as requests.JSONDecodeError
            # This aliases json.JSONDecodeError and simplejson.JSONDecodeError
>           raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)
E           requests.exceptions.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/requests/models.py:975: JSONDecodeError
```

Failed *(hide details)*	test_airline_delays.py::test_year_between2005and2009_distance_lessthan1000miles	0.15	

```
self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
```

Result	Test	Duration	Links

```
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
>           return complexjson.loads(self.text, **kwargs)

venv/lib/python3.9/site-packages/requests/models.py:971:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
encoding = None, cls = None, object_hook = None, parse_float = None, parse_int = None, parse_constant = None, object_pairs_hook = None
use_decimal = False, kw = {}

    def loads(s, encoding=None, cls=None, object_hook=None, parse_float=None,
            parse_int=None, parse_constant=None, object_pairs_hook=None,
            use_decimal=False, **kw):
        """Deserialize ``s`` (a ``str`` or ``unicode`` instance containing a JSON
        document) to a Python object.

        *encoding* determines the encoding used to interpret any
        :class:`bytes` objects decoded by this instance (``'utf-8'`` by
        default). It has no effect when decoding :class:`unicode` objects.

        *object_hook*, if specified, will be called with the result of every
        JSON object decoded and its return value will be used in place of the
        given :class:`dict`.  This can be used to provide custom
        deserializations (e.g. to support JSON-RPC class hinting).

        *object_pairs_hook* is an optional function that will be called with
        the result of any object literal decode with an ordered list of pairs.
        The return value of *object_pairs_hook* will be used instead of the
        :class:`dict`.  This feature can be used to implement custom decoders
        that rely on the order that the key and value pairs are decoded (for
        example, :func:`collections.OrderedDict` will remember the order of
        insertion). If *object_hook* is also defined, the *object_pairs_hook*
        takes priority.

        *parse_float*, if specified, will be called with the string of every
        JSON float to be decoded.  By default, this is equivalent to
        ``float(num_str)``. This can be used to use another datatype or parser
        for JSON floats (e.g. :class:`decimal.Decimal`).

        *parse_int*, if specified, will be called with the string of every
        JSON int to be decoded.  By default, this is equivalent to
```

Result	Test	Duration	Links

```
        ``int(num_str)``.  This can be used to use another datatype or parser
        for JSON integers (e.g. :class:`float`).

        *parse_constant*, if specified, will be called with one of the
        following strings: ``'-Infinity'``, ``'Infinity'``, ``'NaN'``.  This
        can be used to raise an exception if invalid JSON numbers are
        encountered.

        If *use_decimal* is true (default: ``False``) then it implies
        parse_float=decimal.Decimal for parity with ``dump``.

        To use a custom ``JSONDecoder`` subclass, specify it with the ``cls``
        kwarg. NOTE: You should use *object_hook* or *object_pairs_hook* instead
        of subclassing whenever possible.

        """
        if (cls is None and encoding is None and object_hook is None and
                parse_int is None and parse_float is None and
                parse_constant is None and object_pairs_hook is None
                and not use_decimal and not kw):
>           return _default_decoder.decode(s)

venv/lib/python3.9/site-packages/simplejson/__init__.py:525:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
_w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def decode(self, s, _w=WHITESPACE.match, _PY3=PY3):
        """Return the Python representation of ``s`` (a ``str`` or ``unicode``
        instance containing a JSON document)

        """
        if _PY3 and isinstance(s, bytes):
            s = str(s, self.encoding)
>       obj, end = self.raw_decode(s)

venv/lib/python3.9/site-packages/simplejson/decoder.py:370:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
idx = 0, _w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def raw_decode(self, s, idx=0, _w=WHITESPACE.match, _PY3=PY3):
        """Decode a JSON document from ``s`` (a ``str`` or ``unicode``
```

Result	Test	Duration	Links

```
        beginning with a JSON document) and return a 2-tuple of the Python
        representation and the index in ``s`` where the document ended.
        Optionally, ``idx`` can be used to specify an offset in ``s`` where
        the JSON document begins.

        This can be used to decode a JSON document from a string that may
        have extraneous data at the end.

        """
        if idx < 0:
            # Ensure that raw_decode bails on negative indexes, the regex
            # would otherwise mask this behavior. #98
            raise JSONDecodeError('Expecting value', s, idx)
        if _PY3 and not isinstance(s, str):
            raise TypeError("Input string must be text, not bytes")
        # strip UTF-8 bom
        if len(s) > idx:
            ord0 = ord(s[idx])
            if ord0 == 0xfeff:
                idx += 1
            elif ord0 == 0xef and s[idx:idx + 3] == '\xef\xbb\xbf':
                idx += 3
>       return self.scan_once(s, idx=_w(s, idx).end())
E       simplejson.errors.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/simplejson/decoder.py:400: JSONDecodeError

During handling of the above exception, another exception occurred:

    def test_year_between2005and2009_distance_lessthan1000miles():

        params = {'o': "BOS",
                  'dst': "IAD",
                  'a': "PT",
                  'yf' : 2005,
                  'yt' : 2009}

        response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                params=params)

        expected_output = {}

>       assert response.json() == expected_output

test_airline_delays.py:167:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <Response [500]>, kwargs = {}
```

Result	Test	Duration	Links

```
    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
            return complexjson.loads(self.text, **kwargs)
        except JSONDecodeError as e:
            # Catch JSON-related errors and raise as requests.JSONDecodeError
            # This aliases json.JSONDecodeError and simplejson.JSONDecodeError
>           raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)
E           requests.exceptions.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/requests/models.py:975: JSONDecodeError
```

Failed *(hide details)*	test_airline_delays.py::test_year_between2005and2009_distance_between1000and4000miles	0.13	

```
self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
```

Result	Test	Duration	Links

```
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
>           return complexjson.loads(self.text, **kwargs)

venv/lib/python3.9/site-packages/requests/models.py:971:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
encoding = None, cls = None, object_hook = None, parse_float = None, parse_int = None, parse_constant = None, object_pairs_hook = None
use_decimal = False, kw = {}

    def loads(s, encoding=None, cls=None, object_hook=None, parse_float=None,
            parse_int=None, parse_constant=None, object_pairs_hook=None,
            use_decimal=False, **kw):
        """Deserialize ``s`` (a ``str`` or ``unicode`` instance containing a JSON
        document) to a Python object.

        *encoding* determines the encoding used to interpret any
        :class:`bytes` objects decoded by this instance (``'utf-8'`` by
        default). It has no effect when decoding :class:`unicode` objects.

        *object_hook*, if specified, will be called with the result of every
        JSON object decoded and its return value will be used in place of the
        given :class:`dict`.  This can be used to provide custom
        deserializations (e.g. to support JSON-RPC class hinting).

        *object_pairs_hook* is an optional function that will be called with
        the result of any object literal decode with an ordered list of pairs.
        The return value of *object_pairs_hook* will be used instead of the
        :class:`dict`.  This feature can be used to implement custom decoders
        that rely on the order that the key and value pairs are decoded (for
        example, :func:`collections.OrderedDict` will remember the order of
```

Result	Test	Duration	Links

```
        insertion). If *object_hook* is also defined, the *object_pairs_hook*
        takes priority.

        *parse_float*, if specified, will be called with the string of every
        JSON float to be decoded.  By default, this is equivalent to
        ``float(num_str)``. This can be used to use another datatype or parser
        for JSON floats (e.g. :class:`decimal.Decimal`).

        *parse_int*, if specified, will be called with the string of every
        JSON int to be decoded.  By default, this is equivalent to
        ``int(num_str)``.  This can be used to use another datatype or parser
        for JSON integers (e.g. :class:`float`).

        *parse_constant*, if specified, will be called with one of the
        following strings: ``'-Infinity'``, ``'Infinity'``, ``'NaN'``.  This
        can be used to raise an exception if invalid JSON numbers are
        encountered.

        If *use_decimal* is true (default: ``False``) then it implies
        parse_float=decimal.Decimal for parity with ``dump``.

        To use a custom ``JSONDecoder`` subclass, specify it with the ``cls``
        kwarg. NOTE: You should use *object_hook* or *object_pairs_hook* instead
        of subclassing whenever possible.

        """
        if (cls is None and encoding is None and object_hook is None and
                parse_int is None and parse_float is None and
                parse_constant is None and object_pairs_hook is None
                and not use_decimal and not kw):
>           return _default_decoder.decode(s)

venv/lib/python3.9/site-packages/simplejson/__init__.py:525:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
_w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def decode(self, s, _w=WHITESPACE.match, _PY3=PY3):
        """Return the Python representation of ``s`` (a ``str`` or ``unicode``
        instance containing a JSON document)

        """
        if _PY3 and isinstance(s, bytes):
            s = str(s, self.encoding)
>       obj, end = self.raw_decode(s)
```

Result	Test	Duration	Links

```
venv/lib/python3.9/site-packages/simplejson/decoder.py:370:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
idx = 0, _w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def raw_decode(self, s, idx=0, _w=WHITESPACE.match, _PY3=PY3):
        """Decode a JSON document from ``s`` (a ``str`` or ``unicode``
        beginning with a JSON document) and return a 2-tuple of the Python
        representation and the index in ``s`` where the document ended.
        Optionally, ``idx`` can be used to specify an offset in ``s`` where
        the JSON document begins.

        This can be used to decode a JSON document from a string that may
        have extraneous data at the end.

        """
        if idx < 0:
            # Ensure that raw_decode bails on negative indexes, the regex
            # would otherwise mask this behavior. #98
            raise JSONDecodeError('Expecting value', s, idx)
        if _PY3 and not isinstance(s, str):
            raise TypeError("Input string must be text, not bytes")
        # strip UTF-8 bom
        if len(s) > idx:
            ord0 = ord(s[idx])
            if ord0 == 0xfeff:
                idx += 1
            elif ord0 == 0xef and s[idx:idx + 3] == '\xef\xbb\xbf':
                idx += 3
>       return self.scan_once(s, idx=_w(s, idx).end())
E       simplejson.errors.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/simplejson/decoder.py:400: JSONDecodeError

During handling of the above exception, another exception occurred:

    def test_year_between2005and2009_distance_between1000and4000miles():

        params = {'o': "LAS",
                  'dst': "JFK",
                  'a': "KS",
                  'yf' : 2005,
                  'yt' : 2009}

        response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                params=params)
```

Result	Test	Duration	Links

```
        expected_output = []

>       assert response.json() == expected_output

test_airline_delays.py:183:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
            return complexjson.loads(self.text, **kwargs)
        except JSONDecodeError as e:
            # Catch JSON-related errors and raise as requests.JSONDecodeError
            # This aliases json.JSONDecodeError and simplejson.JSONDecodeError
>           raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)
E           requests.exceptions.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/requests/models.py:975: JSONDecodeError
```

Failed *(hide details)*	test_airline_delays.py::test_year_between2005and2009_distance_morethan4000miles	0.12	

```
self = <Response [500]>, kwargs = {}
```

Result	Test	Duration	Links

```
    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
>           return complexjson.loads(self.text, **kwargs)

venv/lib/python3.9/site-packages/requests/models.py:971:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
encoding = None, cls = None, object_hook = None, parse_float = None, parse_int = None, parse_constant = None, object_pairs_hook = None
use_decimal = False, kw = {}

    def loads(s, encoding=None, cls=None, object_hook=None, parse_float=None,
            parse_int=None, parse_constant=None, object_pairs_hook=None,
            use_decimal=False, **kw):
        """Deserialize ``s`` (a ``str`` or ``unicode`` instance containing a JSON
        document) to a Python object.

        *encoding* determines the encoding used to interpret any
        :class:`bytes` objects decoded by this instance (``'utf-8'`` by
        default). It has no effect when decoding :class:`unicode` objects.

        *object_hook*, if specified, will be called with the result of every
```

Result	Test	Duration	Links

```
        JSON object decoded and its return value will be used in place of the
        given :class:`dict`.  This can be used to provide custom
        deserializations (e.g. to support JSON-RPC class hinting).

        *object_pairs_hook* is an optional function that will be called with
        the result of any object literal decode with an ordered list of pairs.
        The return value of *object_pairs_hook* will be used instead of the
        :class:`dict`.  This feature can be used to implement custom decoders
        that rely on the order that the key and value pairs are decoded (for
        example, :func:`collections.OrderedDict` will remember the order of
        insertion). If *object_hook* is also defined, the *object_pairs_hook*
        takes priority.

        *parse_float*, if specified, will be called with the string of every
        JSON float to be decoded.  By default, this is equivalent to
        ``float(num_str)``. This can be used to use another datatype or parser
        for JSON floats (e.g. :class:`decimal.Decimal`).

        *parse_int*, if specified, will be called with the string of every
        JSON int to be decoded.  By default, this is equivalent to
        ``int(num_str)``.  This can be used to use another datatype or parser
        for JSON integers (e.g. :class:`float`).

        *parse_constant*, if specified, will be called with one of the
        following strings: ``'-Infinity'``, ``'Infinity'``, ``'NaN'``.  This
        can be used to raise an exception if invalid JSON numbers are
        encountered.

        If *use_decimal* is true (default: ``False``) then it implies
        parse_float=decimal.Decimal for parity with ``dump``.

        To use a custom ``JSONDecoder`` subclass, specify it with the ``cls``
        kwarg. NOTE: You should use *object_hook* or *object_pairs_hook* instead
        of subclassing whenever possible.

        """
        if (cls is None and encoding is None and object_hook is None and
                parse_int is None and parse_float is None and
                parse_constant is None and object_pairs_hook is None
                and not use_decimal and not kw):
>           return _default_decoder.decode(s)

venv/lib/python3.9/site-packages/simplejson/__init__.py:525:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
_w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True
```

Result	Test	Duration	Links

```
    def decode(self, s, _w=WHITESPACE.match, _PY3=PY3):
        """Return the Python representation of ``s`` (a ``str`` or ``unicode``
        instance containing a JSON document)

        """
        if _PY3 and isinstance(s, bytes):
            s = str(s, self.encoding)
>       obj, end = self.raw_decode(s)

venv/lib/python3.9/site-packages/simplejson/decoder.py:370:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
idx = 0, _w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def raw_decode(self, s, idx=0, _w=WHITESPACE.match, _PY3=PY3):
        """Decode a JSON document from ``s`` (a ``str`` or ``unicode``
        beginning with a JSON document) and return a 2-tuple of the Python
        representation and the index in ``s`` where the document ended.
        Optionally, ``idx`` can be used to specify an offset in ``s`` where
        the JSON document begins.

        This can be used to decode a JSON document from a string that may
        have extraneous data at the end.

        """
        if idx < 0:
            # Ensure that raw_decode bails on negative indexes, the regex
            # would otherwise mask this behavior. #98
            raise JSONDecodeError('Expecting value', s, idx)
        if _PY3 and not isinstance(s, str):
            raise TypeError("Input string must be text, not bytes")
        # strip UTF-8 bom
        if len(s) > idx:
            ord0 = ord(s[idx])
            if ord0 == 0xfeff:
                idx += 1
            elif ord0 == 0xef and s[idx:idx + 3] == '\xef\xbb\xbf':
                idx += 3
>       return self.scan_once(s, idx=_w(s, idx).end())
E       simplejson.errors.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/simplejson/decoder.py:400: JSONDecodeError

During handling of the above exception, another exception occurred:
```

Result	Test	Duration	Links

```
    def test_year_between2005and2009_distance_morethan4000miles():

        params = {'o': "OGG",
                  'dst': "ORD",
                  'a': "YV",
                  'yf' : 2005,
                  'yt' : 2009}

        response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                params=params)

        expected_output = {}

>       assert response.json() == expected_output

test_airline_delays.py:199:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
            return complexjson.loads(self.text, **kwargs)
        except JSONDecodeError as e:
```

Result	Test	Duration	Links

```
            # Catch JSON-related errors and raise as requests.JSONDecodeError
            # This aliases json.JSONDecodeError and simplejson.JSONDecodeError
>           raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)
E           requests.exceptions.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/requests/models.py:975: JSONDecodeError
```

Failed *(hide details)*	test_airline_delays.py::test_year_between2010and2014_distance_lessthan1000miles	0.52	

```
self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
>           return complexjson.loads(self.text, **kwargs)

venv/lib/python3.9/site-packages/requests/models.py:971:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
encoding = None, cls = None, object_hook = None, parse_float = None, parse_int = None, parse_constant = None, object_pairs_hook = None
use_decimal = False, kw = {}

    def loads(s, encoding=None, cls=None, object_hook=None, parse_float=None,
```

Result	Test	Duration	Links

```
        parse_int=None, parse_constant=None, object_pairs_hook=None,
        use_decimal=False, **kw):
    """Deserialize ``s`` (a ``str`` or ``unicode`` instance containing a JSON
    document) to a Python object.

    *encoding* determines the encoding used to interpret any
    :class:`bytes` objects decoded by this instance (``'utf-8'`` by
    default). It has no effect when decoding :class:`unicode` objects.

    *object_hook*, if specified, will be called with the result of every
    JSON object decoded and its return value will be used in place of the
    given :class:`dict`.  This can be used to provide custom
    deserializations (e.g. to support JSON-RPC class hinting).

    *object_pairs_hook* is an optional function that will be called with
    the result of any object literal decode with an ordered list of pairs.
    The return value of *object_pairs_hook* will be used instead of the
    :class:`dict`.  This feature can be used to implement custom decoders
    that rely on the order that the key and value pairs are decoded (for
    example, :func:`collections.OrderedDict` will remember the order of
    insertion). If *object_hook* is also defined, the *object_pairs_hook*
    takes priority.

    *parse_float*, if specified, will be called with the string of every
    JSON float to be decoded.  By default, this is equivalent to
    ``float(num_str)``. This can be used to use another datatype or parser
    for JSON floats (e.g. :class:`decimal.Decimal`).

    *parse_int*, if specified, will be called with the string of every
    JSON int to be decoded.  By default, this is equivalent to
    ``int(num_str)``.  This can be used to use another datatype or parser
    for JSON integers (e.g. :class:`float`).

    *parse_constant*, if specified, will be called with one of the
    following strings: ``'-Infinity'``, ``'Infinity'``, ``'NaN'``.  This
    can be used to raise an exception if invalid JSON numbers are
    encountered.

    If *use_decimal* is true (default: ``False``) then it implies
    parse_float=decimal.Decimal for parity with ``dump``.

    To use a custom ``JSONDecoder`` subclass, specify it with the ``cls``
    kwarg. NOTE: You should use *object_hook* or *object_pairs_hook* instead
    of subclassing whenever possible.

    """
    if (cls is None and encoding is None and object_hook is None and
            parse_int is None and parse_float is None and
            parse_constant is None and object_pairs_hook is None
```

Result	Test	Duration	Links

```
                and not use_decimal and not kw):
>           return _default_decoder.decode(s)

venv/lib/python3.9/site-packages/simplejson/__init__.py:525:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
_w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def decode(self, s, _w=WHITESPACE.match, _PY3=PY3):
        """Return the Python representation of ``s`` (a ``str`` or ``unicode``
        instance containing a JSON document)

        """
        if _PY3 and isinstance(s, bytes):
            s = str(s, self.encoding)
>       obj, end = self.raw_decode(s)

venv/lib/python3.9/site-packages/simplejson/decoder.py:370:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
idx = 0, _w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def raw_decode(self, s, idx=0, _w=WHITESPACE.match, _PY3=PY3):
        """Decode a JSON document from ``s`` (a ``str`` or ``unicode``
        beginning with a JSON document) and return a 2-tuple of the Python
        representation and the index in ``s`` where the document ended.
        Optionally, ``idx`` can be used to specify an offset in ``s`` where
        the JSON document begins.

        This can be used to decode a JSON document from a string that may
        have extraneous data at the end.

        """
        if idx < 0:
            # Ensure that raw_decode bails on negative indexes, the regex
            # would otherwise mask this behavior. #98
            raise JSONDecodeError('Expecting value', s, idx)
        if _PY3 and not isinstance(s, str):
            raise TypeError("Input string must be text, not bytes")
        # strip UTF-8 bom
        if len(s) > idx:
            ord0 = ord(s[idx])
            if ord0 == 0xfeff:
```

Result	Test	Duration	Links

```
                idx += 1
            elif ord0 == 0xef and s[idx:idx + 3] == '\xef\xbb\xbf':
                idx += 3
>       return self.scan_once(s, idx=_w(s, idx).end())
E       simplejson.errors.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/simplejson/decoder.py:400: JSONDecodeError

During handling of the above exception, another exception occurred:

    def test_year_between2010and2014_distance_lessthan1000miles():

        params = {'o': "ORD",
                  'dst': "DEN",
                  'a': "B6",
                  'yf' : 2010,
                  'yt' : 2014}

        response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                params=params)

        expected_output = {}

>       assert response.json() == expected_output

test_airline_delays.py:215:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
```

Result	Test	Duration	Links

```
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
            return complexjson.loads(self.text, **kwargs)
        except JSONDecodeError as e:
            # Catch JSON-related errors and raise as requests.JSONDecodeError
            # This aliases json.JSONDecodeError and simplejson.JSONDecodeError
>           raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)
E           requests.exceptions.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/requests/models.py:975: JSONDecodeError
```

Failed *(hide details)*	test_airline_delays.py::test_year_between2010and2014_distance_between1000and4000miles	0.12	

```
self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
>           return complexjson.loads(self.text, **kwargs)
```

Result	Test	Duration	Links

```
venv/lib/python3.9/site-packages/requests/models.py:971:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
encoding = None, cls = None, object_hook = None, parse_float = None, parse_int = None, parse_constant = None, object_pairs_hook = None
use_decimal = False, kw = {}

    def loads(s, encoding=None, cls=None, object_hook=None, parse_float=None,
            parse_int=None, parse_constant=None, object_pairs_hook=None,
            use_decimal=False, **kw):
        """Deserialize ``s`` (a ``str`` or ``unicode`` instance containing a JSON
        document) to a Python object.

        *encoding* determines the encoding used to interpret any
        :class:`bytes` objects decoded by this instance (``'utf-8'`` by
        default). It has no effect when decoding :class:`unicode` objects.

        *object_hook*, if specified, will be called with the result of every
        JSON object decoded and its return value will be used in place of the
        given :class:`dict`.  This can be used to provide custom
        deserializations (e.g. to support JSON-RPC class hinting).

        *object_pairs_hook* is an optional function that will be called with
        the result of any object literal decode with an ordered list of pairs.
        The return value of *object_pairs_hook* will be used instead of the
        :class:`dict`.  This feature can be used to implement custom decoders
        that rely on the order that the key and value pairs are decoded (for
        example, :func:`collections.OrderedDict` will remember the order of
        insertion). If *object_hook* is also defined, the *object_pairs_hook*
        takes priority.

        *parse_float*, if specified, will be called with the string of every
        JSON float to be decoded.  By default, this is equivalent to
        ``float(num_str)``. This can be used to use another datatype or parser
        for JSON floats (e.g. :class:`decimal.Decimal`).

        *parse_int*, if specified, will be called with the string of every
        JSON int to be decoded.  By default, this is equivalent to
        ``int(num_str)``.  This can be used to use another datatype or parser
        for JSON integers (e.g. :class:`float`).

        *parse_constant*, if specified, will be called with one of the
        following strings: ``'-Infinity'``, ``'Infinity'``, ``'NaN'``.  This
        can be used to raise an exception if invalid JSON numbers are
        encountered.

        If *use_decimal* is true (default: ``False``) then it implies
```

Result	Test	Duration	Links

```
        parse_float=decimal.Decimal for parity with ``dump``.

        To use a custom ``JSONDecoder`` subclass, specify it with the ``cls``
        kwarg. NOTE: You should use *object_hook* or *object_pairs_hook* instead
        of subclassing whenever possible.

        """
        if (cls is None and encoding is None and object_hook is None and
                parse_int is None and parse_float is None and
                parse_constant is None and object_pairs_hook is None
                and not use_decimal and not kw):
>           return _default_decoder.decode(s)

venv/lib/python3.9/site-packages/simplejson/__init__.py:525:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
_w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def decode(self, s, _w=WHITESPACE.match, _PY3=PY3):
        """Return the Python representation of ``s`` (a ``str`` or ``unicode``
        instance containing a JSON document)

        """
        if _PY3 and isinstance(s, bytes):
            s = str(s, self.encoding)
>       obj, end = self.raw_decode(s)

venv/lib/python3.9/site-packages/simplejson/decoder.py:370:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
idx = 0, _w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def raw_decode(self, s, idx=0, _w=WHITESPACE.match, _PY3=PY3):
        """Decode a JSON document from ``s`` (a ``str`` or ``unicode``
        beginning with a JSON document) and return a 2-tuple of the Python
        representation and the index in ``s`` where the document ended.
        Optionally, ``idx`` can be used to specify an offset in ``s`` where
        the JSON document begins.

        This can be used to decode a JSON document from a string that may
        have extraneous data at the end.

        """
```

Result	Test	Duration	Links

```
        if idx < 0:
            # Ensure that raw_decode bails on negative indexes, the regex
            # would otherwise mask this behavior. #98
            raise JSONDecodeError('Expecting value', s, idx)
        if _PY3 and not isinstance(s, str):
            raise TypeError("Input string must be text, not bytes")
        # strip UTF-8 bom
        if len(s) > idx:
            ord0 = ord(s[idx])
            if ord0 == 0xfeff:
                idx += 1
            elif ord0 == 0xef and s[idx:idx + 3] == '\xef\xbb\xbf':
                idx += 3
>       return self.scan_once(s, idx=_w(s, idx).end())
E       simplejson.errors.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/simplejson/decoder.py:400: JSONDecodeError

During handling of the above exception, another exception occurred:

    def test_year_between2010and2014_distance_between1000and4000miles():

        params = {'o': "ATL",
                  'dst': "SAN",
                  'a': "9E",
                  'yf' : 2010,
                  'yt' : 2014}

        response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                params=params)

        expected_output = {}

>       assert response.json() == expected_output

test_airline_delays.py:231:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
```

Result	Test	Duration	Links

```
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
            return complexjson.loads(self.text, **kwargs)
        except JSONDecodeError as e:
            # Catch JSON-related errors and raise as requests.JSONDecodeError
            # This aliases json.JSONDecodeError and simplejson.JSONDecodeError
>           raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)
E           requests.exceptions.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/requests/models.py:975: JSONDecodeError
```

Failed *(hide details)*	test_airline_delays.py::test_year_between2010and2014_distance_morethan4000miles	0.14	

```
self = <Response [500]>, kwargs = {}

    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
```

Result	Test	Duration	Links

```
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
>           return complexjson.loads(self.text, **kwargs)

venv/lib/python3.9/site-packages/requests/models.py:971:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
encoding = None, cls = None, object_hook = None, parse_float = None, parse_int = None, parse_constant = None, object_pairs_hook = None
use_decimal = False, kw = {}

    def loads(s, encoding=None, cls=None, object_hook=None, parse_float=None,
            parse_int=None, parse_constant=None, object_pairs_hook=None,
            use_decimal=False, **kw):
        """Deserialize ``s`` (a ``str`` or ``unicode`` instance containing a JSON
        document) to a Python object.

        *encoding* determines the encoding used to interpret any
        :class:`bytes` objects decoded by this instance (``'utf-8'`` by
        default). It has no effect when decoding :class:`unicode` objects.

        *object_hook*, if specified, will be called with the result of every
        JSON object decoded and its return value will be used in place of the
        given :class:`dict`.  This can be used to provide custom
        deserializations (e.g. to support JSON-RPC class hinting).

        *object_pairs_hook* is an optional function that will be called with
        the result of any object literal decode with an ordered list of pairs.
        The return value of *object_pairs_hook* will be used instead of the
        :class:`dict`.  This feature can be used to implement custom decoders
        that rely on the order that the key and value pairs are decoded (for
        example, :func:`collections.OrderedDict` will remember the order of
        insertion). If *object_hook* is also defined, the *object_pairs_hook*
        takes priority.

        *parse_float*, if specified, will be called with the string of every
        JSON float to be decoded.  By default, this is equivalent to
        ``float(num_str)``. This can be used to use another datatype or parser
        for JSON floats (e.g. :class:`decimal.Decimal`).

        *parse_int*, if specified, will be called with the string of every
```

Result | Test | Duration | Links

```
        JSON int to be decoded.  By default, this is equivalent to
        ``int(num_str)``.  This can be used to use another datatype or parser
        for JSON integers (e.g. :class:`float`).

        *parse_constant*, if specified, will be called with one of the
        following strings: ``'-Infinity'``, ``'Infinity'``, ``'NaN'``.  This
        can be used to raise an exception if invalid JSON numbers are
        encountered.

        If *use_decimal* is true (default: ``False``) then it implies
        parse_float=decimal.Decimal for parity with ``dump``.

        To use a custom ``JSONDecoder`` subclass, specify it with the ``cls``
        kwarg. NOTE: You should use *object_hook* or *object_pairs_hook* instead
        of subclassing whenever possible.

        """
        if (cls is None and encoding is None and object_hook is None and
                parse_int is None and parse_float is None and
                parse_constant is None and object_pairs_hook is None
                and not use_decimal and not kw):
>           return _default_decoder.decode(s)

venv/lib/python3.9/site-packages/simplejson/__init__.py:525:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
_w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def decode(self, s, _w=WHITESPACE.match, _PY3=PY3):
        """Return the Python representation of ``s`` (a ``str`` or ``unicode``
        instance containing a JSON document)

        """
        if _PY3 and isinstance(s, bytes):
            s = str(s, self.encoding)
>       obj, end = self.raw_decode(s)

venv/lib/python3.9/site-packages/simplejson/decoder.py:370:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <simplejson.decoder.JSONDecoder object at 0x7fb04e88f850>
s = "<!doctype html>\n<html lang=en>\n  <head>\n    <title>TypeError: 'NoneType' object is not iterable\n // Werkzeug Debu...in
airline_delays\n    json_obj=dict(zip(row_headers,result))\nTypeError: 'NoneType' object is not iterable\n\n\n-->\n"
idx = 0, _w = <built-in method match of re.Pattern object at 0x7fb04e05d3f0>, _PY3 = True

    def raw_decode(self, s, idx=0, _w=WHITESPACE.match, _PY3=PY3):
```

Result | Test | Duration | Links

```
        """Decode a JSON document from ``s`` (a ``str`` or ``unicode``
        beginning with a JSON document) and return a 2-tuple of the Python
        representation and the index in ``s`` where the document ended.
        Optionally, ``idx`` can be used to specify an offset in ``s`` where
        the JSON document begins.

        This can be used to decode a JSON document from a string that may
        have extraneous data at the end.

        """
        if idx < 0:
            # Ensure that raw_decode bails on negative indexes, the regex
            # would otherwise mask this behavior. #98
            raise JSONDecodeError('Expecting value', s, idx)
        if _PY3 and not isinstance(s, str):
            raise TypeError("Input string must be text, not bytes")
        # strip UTF-8 bom
        if len(s) > idx:
            ord0 = ord(s[idx])
            if ord0 == 0xfeff:
                idx += 1
            elif ord0 == 0xef and s[idx:idx + 3] == '\xef\xbb\xbf':
                idx += 3
>       return self.scan_once(s, idx=_w(s, idx).end())
E       simplejson.errors.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/simplejson/decoder.py:400: JSONDecodeError

During handling of the above exception, another exception occurred:

    def test_year_between2010and2014_distance_morethan4000miles():

        params = {'o': "BOS",
                  'dst': "HNL",
                  'a': "HA",
                  'yf' : 2010,
                  'yt' : 2014}

        response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                params=params)

        expected_output = {}

>       assert response.json() == expected_output

test_airline_delays.py:247:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <Response [500]>, kwargs = {}
```

Result	Test	Duration	Links

```
    def json(self, **kwargs):
        r"""Returns the json-encoded content of a response, if any.

        :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
        :raises requests.exceptions.JSONDecodeError: If the response body does not
            contain valid json.
        """

        if not self.encoding and self.content and len(self.content) > 3:
            # No encoding set. JSON RFC 4627 section 3 states we should expect
            # UTF-8, -16 or -32. Detect which one to use; If the detection or
            # decoding fails, fall back to `self.text` (using charset_normalizer to make
            # a best guess).
            encoding = guess_json_utf(self.content)
            if encoding is not None:
                try:
                    return complexjson.loads(self.content.decode(encoding), **kwargs)
                except UnicodeDecodeError:
                    # Wrong UTF codec detected; usually because it's not UTF-8
                    # but some other 8-bit codec.  This is an RFC violation,
                    # and the server didn't bother to tell us what codec *was*
                    # used.
                    pass
                except JSONDecodeError as e:
                    raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

        try:
            return complexjson.loads(self.text, **kwargs)
        except JSONDecodeError as e:
            # Catch JSON-related errors and raise as requests.JSONDecodeError
            # This aliases json.JSONDecodeError and simplejson.JSONDecodeError
>           raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)
E           requests.exceptions.JSONDecodeError: Expecting value: line 1 column 1 (char 0)

venv/lib/python3.9/site-packages/requests/models.py:975: JSONDecodeError
```

Failed *(hide details)*	test_airline_delays.py::test_invalid_year	3.55	

Result	Test	Duration	Links

```
def test_invalid_year():

        params = {'o': "SAT",
                  'dst': "PHX",
                  'a': "AA",
                  'yf' : 1985,
                  'yt' : 2030}

        with pytest.raises(ValueError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                    params=params)
E           Failed: DID NOT RAISE <class 'ValueError'>

test_airline_delays.py:361: Failed
```

Failed *(hide details)*	test_airline_delays.py::test_invalid_airport	3.32	

```
def test_invalid_airport():

        params = {'o': "BBB",
                  'dst': "CCC",
                  'a': "AA",
                  'yf' : 1990,
                  'yt' : 2021}

        with pytest.raises(ValueError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                    params=params)
E           Failed: DID NOT RAISE <class 'ValueError'>

test_airline_delays.py:374: Failed
```

Failed *(hide details)*	test_airline_delays.py::test_invalid_flight	3.26	

Result	Test	Duration	Links

```
def test_invalid_flight():

        params = {'o': "SAT",
                  'dst': "SAT",
                  'a': "AA",
                  'yf' : 1990,
                  'yt' : 2021}

        with pytest.raises(ValueError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                    params=params)
E           Failed: DID NOT RAISE <class 'ValueError'>

test_airline_delays.py:387: Failed
```

Failed *(hide details)*	test_airline_delays.py::test_invalid_airline	2.96	

```
def test_invalid_airline():

        params = {'o': "SAT",
                  'dst': "PHX",
                  'a': "NOT",
                  'yf' : 1990,
                  'yt' : 2021}

        with pytest.raises(ValueError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                    params=params)
E           Failed: DID NOT RAISE <class 'ValueError'>

test_airline_delays.py:400: Failed
```

Failed *(hide details)*	test_airline_delays.py::test_invalid_month	0.12	

Result	Test	Duration	Links

```
def test_invalid_month():

        params = {'o': "SAT",
                  'dst': "PHX",
                  'a': "AA",
                  'yf' : 1990,
                  'yt' : 2021,
                  'm' : 20}

        with pytest.raises(ValueError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                    params=params)
E           Failed: DID NOT RAISE <class 'ValueError'>

test_airline_delays.py:414: Failed
```

Failed *(hide details)*	test_airline_delays.py::test_invalid_day	0.13	

```
def test_invalid_day():

        params = {'o': "SAT",
                  'dst': "PHX",
                  'a': "AA",
                  'yf' : 1990,
                  'yt' : 2021,
                  'm' : 12,
                  'd' : -1}

        with pytest.raises(ValueError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                    params=params)
E           Failed: DID NOT RAISE <class 'ValueError'>

test_airline_delays.py:429: Failed
```

Failed *(hide details)*	test_airline_delays.py::test_illegal_parameters	2.94	

Result	Test	Duration	Links

```
def test_illegal_parameters():

        params = {'o': "SAT",
                  'dst': "PHX",
                  'a': "AA"}

        with pytest.raises(TypeError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                    params=params)
E           Failed: DID NOT RAISE <class 'TypeError'>

test_airline_delays.py:440: Failed
```

Failed *(hide details)*	test_airline_delays.py::test_yearFrom_greater_than_yearTo	0.10	

```
def test_yearFrom_greater_than_yearTo():

        params = {'o': "SAT",
                  'dst': "PHX",
                  'a': "AA",
                  'yf' : 2021,
                  'yt' : 1990}

        with pytest.raises(ValueError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/airline_delays",
                    params=params)
E           Failed: DID NOT RAISE <class 'ValueError'>

test_airline_delays.py:453: Failed
```

Failed *(hide details)*	test_airline_stats.py::test_invalid_year	4.37	

```
def test_invalid_year():

        params = {'o': "ORD",
                  'dst': "OGG",
                  'yf' : 1990,
                  'yt' : 2030}

        with pytest.raises(ValueError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/airlines_stats",
                    params=params)
E           Failed: DID NOT RAISE <class 'ValueError'>

test_airline_stats.py:348: Failed
```

Failed *(hide details)*	test_airline_stats.py::test_invalid_airport	4.42	

Result	Test	Duration	Links

```
def test_invalid_airport():

        params = {'o': "AAA",
                  'dst': "OGG",
                  'yf' : 1990,
                  'yt' : 2021}

        with pytest.raises(ValueError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/airlines_stats",
                    params=params)
E           Failed: DID NOT RAISE <class 'ValueError'>

test_airline_stats.py:360: Failed
```

Failed *(hide details)*	test_airline_stats.py::test_invalid_flight	4.81	

```
def test_invalid_flight():

        params = {'o': "BOS",
                  'dst': "BOS",
                  'yf' : 1990,
                  'yt' : 2021}

        with pytest.raises(ValueError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/airlines_stats",
                    params=params)
E           Failed: DID NOT RAISE <class 'ValueError'>

test_airline_stats.py:372: Failed
```

Failed *(hide details)*	test_airline_stats.py::test_invalid_month	0.09	

```
def test_invalid_month():

        params = {'o': "SAT",
                  'dst': "PHX",
                  'yf' : 1990,
                  'yt' : 2021,
                  'm': 14}

        with pytest.raises(ValueError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/airlines_stats",
                    params=params)
E           Failed: DID NOT RAISE <class 'ValueError'>

test_airline_stats.py:385: Failed
```

Result	Test	Duration	Links
Failed *(hide details)*	test_airline_stats.py::test_invalid_day	0.10	

```
def test_invalid_day():

        params = {'o': "SAT",
                  'dst': "PHX",
                  'yf' : 1990,
                  'yt' : 2021,
                  'm': 12,
                  'd': 40}

        with pytest.raises(ValueError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/airlines_stats",
                    params=params)
E           Failed: DID NOT RAISE <class 'ValueError'>

test_airline_stats.py:400: Failed
```

Failed *(hide details)*	test_airline_stats.py::test_illegal_parameters	2.40	

```
def test_illegal_parameters():

        params = {'o': "SAT",
                  'dst': "PHX"}

        with pytest.raises(TypeError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/airlines_stats",
                    params=params)
E           Failed: DID NOT RAISE <class 'TypeError'>

test_airline_stats.py:411: Failed
```

Failed *(hide details)*	test_airline_stats.py::test_yearFrom_greater_than_yearTo	0.13	

```
def test_yearFrom_greater_than_yearTo():

        params = {'o': "SAT",
                  'dst': "PHX",
                  'yf' : 2021,
                  'yt' : 1990}

        with pytest.raises(ValueError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/airlines_stats",
                    params=params)
E           Failed: DID NOT RAISE <class 'ValueError'>

test_airline_stats.py:424: Failed
```

Result	Test	Duration	Links
Failed *(hide details)*	test_delays_comparison.py::test_year_between1990and1994_distance_lessthan1000miles	0.29	

```
def test_year_between1990and1994_distance_lessthan1000miles():

        params = {'o': "SAT",
                  'dst': "PHX",
                  'a': "VX",
                  'yf' : 1990,
                  'yt' : 1995}

        response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                params=params)

        expected_output = []

>       assert response.json() == expected_output
E       AssertionError: assert [{'carrier': ...': None, ...}] == []
E         Left contains one more item: {'carrier': None, 'late_aircraft': None, 'nas': None, 'sec': None, ...}
E         Use -v to get more diff

test_delays_comparison.py:20: AssertionError
```

Failed *(hide details)*	test_delays_comparison.py::test_year_between1990and1994_distance_between1000and4000miles	0.22	

```
def test_year_between1990and1994_distance_between1000and4000miles():

        params = {'o': "SFO",
                  'dst': "EWR",
                  'a': "US",
                  'yf' : 1990,
                  'yt' : 1994}

        response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                params=params)

        expected_output = []

>       assert response.json() == expected_output
E       AssertionError: assert [{'carrier': ...': None, ...}] == []
E         Left contains one more item: {'carrier': None, 'late_aircraft': None, 'nas': None, 'sec': None, ...}
E         Use -v to get more diff

test_delays_comparison.py:37: AssertionError
```

Failed *(hide details)*	test_delays_comparison.py::test_year_between1990and1994_distance_morethan4000miles	0.24	

Result	Test	Duration	Links

```
def test_year_between1990and1994_distance_morethan4000miles():

        params = {'o': "HNL",
                  'dst': "JFK",
                  'a': "DL",
                  'yf' : 1990,
                  'yt' : 1994}

        response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                params=params)

        expected_output = []

>       assert response.json() == expected_output
E       AssertionError: assert [{'carrier': ...': None, ...}] == []
E         Left contains one more item: {'carrier': None, 'late_aircraft': None, 'nas': None, 'sec': None, ...}
E         Use -v to get more diff

test_delays_comparison.py:53: AssertionError
```

Failed *(hide details)*	test_delays_comparison.py::test_year_between1995and1999_distance_lessthan1000miles	0.21	

```
def test_year_between1995and1999_distance_lessthan1000miles():

        params = {'o': "LGA",
                  'dst': "ORD",
                  'a': "AX",
                  'yf' : 1995,
                  'yt' : 1999}

        response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                params=params)

        expected_output = []

>       assert response.json() == expected_output
E       AssertionError: assert [{'carrier': ...': None, ...}] == []
E         Left contains one more item: {'carrier': None, 'late_aircraft': None, 'nas': None, 'sec': None, ...}
E         Use -v to get more diff

test_delays_comparison.py:69: AssertionError
```

Failed *(hide details)*	test_delays_comparison.py::test_year_between1995and1999_distance_between1000and4000miles	0.25	

Result	Test	Duration	Links

```
def test_year_between1995and1999_distance_between1000and4000miles():

        params = {'o': "IAH",
                  'dst': "SEA",
                  'a': "NK",
                  'yf' : 1995,
                  'yt' : 1999}

        response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                params=params)

        expected_output = []

>       assert response.json() == expected_output
E       AssertionError: assert [{'carrier': ...': None, ...}] == []
E         Left contains one more item: {'carrier': None, 'late_aircraft': None, 'nas': None, 'sec': None, ...}
E         Use -v to get more diff

test_delays_comparison.py:85: AssertionError
```

Failed *(hide details)*	test_delays_comparison.py::test_year_between1995and1999_distance_morethan4000miles	0.21	

```
def test_year_between1995and1999_distance_morethan4000miles():

        params = {'o': "HNL",
                  'dst': "ATL",
                  'a': "EM",
                  'yf' : 1995,
                  'yt' : 1999}

        response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                params=params)

        expected_output = []

>       assert response.json() == expected_output
E       AssertionError: assert [{'carrier': ...': None, ...}] == []
E         Left contains one more item: {'carrier': None, 'late_aircraft': None, 'nas': None, 'sec': None, ...}
E         Use -v to get more diff

test_delays_comparison.py:101: AssertionError
```

Failed *(hide details)*	test_delays_comparison.py::test_year_between2000and2004_distance_lessthan1000miles	0.22	

Result	Test	Duration	Links

```
def test_year_between2000and2004_distance_lessthan1000miles():

        params = {'o': "DFW",
                  'dst': "MKE",
                  'a': "G7",
                  'yf' : 2000,
                  'yt' : 2004}

        response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                params=params)

        expected_output = []

>       assert response.json() == expected_output
E       AssertionError: assert [{'carrier': ...': None, ...}] == []
E         Left contains one more item: {'carrier': None, 'late_aircraft': None, 'nas': None, 'sec': None, ...}
E         Use -v to get more diff

test_delays_comparison.py:117: AssertionError
```

Failed *(hide details)*	test_delays_comparison.py::test_year_between2000and2004_distance_between1000and4000miles	0.24	

```
def test_year_between2000and2004_distance_between1000and4000miles():

        params = {'o': "HNL",
                  'dst': "LAX",
                  'a': "YX",
                  'yf' : 2000,
                  'yt' : 2004}

        response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                params=params)

        expected_output = []

>       assert response.json() == expected_output
E       AssertionError: assert [{'carrier': ...': None, ...}] == []
E         Left contains one more item: {'carrier': None, 'late_aircraft': None, 'nas': None, 'sec': None, ...}
E         Use -v to get more diff

test_delays_comparison.py:133: AssertionError
```

Failed *(hide details)*	test_delays_comparison.py::test_year_between2000and2004_distance_morethan4000miles	1.22	

Result	Test	Duration	Links

```
def test_year_between2000and2004_distance_morethan4000miles():

        params = {'o': "IAH",
                  'dst': "HNL",
                  'a': "OH",
                  'yf' : 2000,
                  'yt' : 2004}

        response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                params=params)

        expected_output = []

>       assert response.json() == expected_output
E       AssertionError: assert [{'carrier': ...': None, ...}] == []
E         Left contains one more item: {'carrier': None, 'late_aircraft': None, 'nas': None, 'sec': None, ...}
E         Use -v to get more diff

test_delays_comparison.py:149: AssertionError
```

Failed *(hide details)*	test_delays_comparison.py::test_year_between2005and2009_distance_lessthan1000miles	0.29	

```
def test_year_between2005and2009_distance_lessthan1000miles():

        params = {'o': "BOS",
                  'dst': "IAD",
                  'a': "PT",
                  'yf' : 2005,
                  'yt' : 2009}

        response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                params=params)

        expected_output = []

>       assert response.json() == expected_output
E       AssertionError: assert [{'carrier': ...': None, ...}] == []
E         Left contains one more item: {'carrier': None, 'late_aircraft': None, 'nas': None, 'sec': None, ...}
E         Use -v to get more diff

test_delays_comparison.py:166: AssertionError
```

Failed *(hide details)*	test_delays_comparison.py::test_year_between2005and2009_distance_between1000and4000miles	0.34	

Result	Test	Duration	Links

```
def test_year_between2005and2009_distance_between1000and4000miles():

        params = {'o': "LAS",
                  'dst': "JFK",
                  'a': "KS",
                  'yf' : 2005,
                  'yt' : 2009}

        response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                params=params)

        expected_output = []

>       assert response.json() == expected_output
E       AssertionError: assert [{'carrier': ...': None, ...}] == []
E         Left contains one more item: {'carrier': None, 'late_aircraft': None, 'nas': None, 'sec': None, ...}
E         Use -v to get more diff

test_delays_comparison.py:182: AssertionError
```

Failed *(hide details)*	test_delays_comparison.py::test_year_between2005and2009_distance_morethan4000miles	0.34	

```
def test_year_between2005and2009_distance_morethan4000miles():

        params = {'o': "OGG",
                  'dst': "ORD",
                  'a': "YV",
                  'yf' : 2005,
                  'yt' : 2009}

        response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                params=params)

        expected_output = []

>       assert response.json() == expected_output
E       AssertionError: assert [{'carrier': ...': None, ...}] == []
E         Left contains one more item: {'carrier': None, 'late_aircraft': None, 'nas': None, 'sec': None, ...}
E         Use -v to get more diff

test_delays_comparison.py:198: AssertionError
```

Failed *(hide details)*	test_delays_comparison.py::test_year_between2010and2014_distance_lessthan1000miles	0.30	

Result	Test	Duration	Links

```
def test_year_between2010and2014_distance_lessthan1000miles():

        params = {'o': "ORD",
                  'dst': "DEN",
                  'a': "B6",
                  'yf' : 2010,
                  'yt' : 2014}

        response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                params=params)

        expected_output = []

>       assert response.json() == expected_output
E       AssertionError: assert [{'carrier': ...': None, ...}] == []
E         Left contains one more item: {'carrier': None, 'late_aircraft': None, 'nas': None, 'sec': None, ...}
E         Use -v to get more diff

test_delays_comparison.py:214: AssertionError
```

Failed *(hide details)*	test_delays_comparison.py::test_year_between2010and2014_distance_between1000and4000miles	0.27	

```
def test_year_between2010and2014_distance_between1000and4000miles():

        params = {'o': "ATL",
                  'dst': "SAN",
                  'a': "9E",
                  'yf' : 2010,
                  'yt' : 2014}

        response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                params=params)

        expected_output = []

>       assert response.json() == expected_output
E       AssertionError: assert [{'carrier': ...': None, ...}] == []
E         Left contains one more item: {'carrier': None, 'late_aircraft': None, 'nas': None, 'sec': None, ...}
E         Use -v to get more diff

test_delays_comparison.py:230: AssertionError
```

Failed *(hide details)*	test_delays_comparison.py::test_year_between2010and2014_distance_morethan4000miles	0.32	

Result	Test	Duration	Links

```
def test_year_between2010and2014_distance_morethan4000miles():

        params = {'o': "BOS",
                  'dst': "HNL",
                  'a': "HA",
                  'yf' : 2010,
                  'yt' : 2014}

        response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                params=params)

        print(response.json())

        expected_output = []

>       assert response.json() == expected_output
E       AssertionError: assert [{'carrier': ...': None, ...}] == []
E         Left contains one more item: {'carrier': None, 'late_aircraft': None, 'nas': None, 'sec': None, ...}
E         Use -v to get more diff

test_delays_comparison.py:248: AssertionError
 ------------------------------Captured stdout call------------------------------
[{'carrier': None, 'nas': None, 'sec': None, 'late_aircraft': None, 'weather': None}]
```

Failed *(hide details)*	test_delays_comparison.py::test_year_between2015and2019_distance_morethan4000miles	16.11	

Result	Test	Duration	Links

```
def test_year_between2015and2019_distance_morethan4000miles():

        params = {'o': "EWR",
                  'dst': "HNL",
                  'a': "UA",
                  'yf' : 2015,
                  'yt' : 2019}

        response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                params=params)

        # should return an array of 2 objects even is 1 airline only, if not front end won't show
        expected_output = [{'carrier': 6.1303, 'nas': 2.713, 'sec': 0.0, 'late_aircraft': 4.0792, 'weather': 0.8791}
        ,{'carrier': 6.1303, 'nas': 2.713, 'sec': 0.0, 'late_aircraft': 4.0792, 'weather': 0.8791}]

>       assert response.json() == expected_output
E       AssertionError: assert [{'carrier': ...c': 0.0, ...}] == [{'carrier': ...c': 0.0, ...}]
E         Right contains one more item: {'carrier': 6.1303, 'late_aircraft': 4.0792, 'nas': 2.713, 'sec': 0.0, ...}
E         Use -v to get more diff

test_delays_comparison.py:303: AssertionError
```

Failed *(hide details)*	test_delays_comparison.py::test_year_between2020and2021_distance_between1000and4000miles	2.96	

```
def test_year_between2020and2021_distance_between1000and4000miles():

        params = {'o': "OAK",
                  'dst': "BWI",
                  'a': "WN",
                  'yf' : 2020,
                  'yt' : 2021}

        response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                params=params)

        # should return an array of 2 objects even is 1 airline only, if not front end won't show
        expected_output = [{'carrier': 1.0153, 'nas': 0.4693, 'sec': 0.0, 'late_aircraft': 0.089, 'weather': 0.0}
        , {'carrier': 1.0153, 'nas': 0.4693, 'sec': 0.0, 'late_aircraft': 0.089, 'weather': 0.0}]

>       assert response.json() == expected_output
E       AssertionError: assert [{'carrier': ...c': 0.0, ...}] == [{'carrier': ...c': 0.0, ...}]
E         Right contains one more item: {'carrier': 1.0153, 'late_aircraft': 0.089, 'nas': 0.4693, 'sec': 0.0, ...}
E         Use -v to get more diff

test_delays_comparison.py:339: AssertionError
```

Failed *(hide details)*	test_delays_comparison.py::test_year_between2020and2021_distance_morethan4000miles	2.72	

Result	Test	Duration	Links

```
def test_year_between2020and2021_distance_morethan4000miles():

        params = {'o': "ORD",
                  'dst': "OGG",
                  'a': "UA",
                  'yf' : 2020,
                  'yt' : 2021}


        response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                params=params)

        # should return an array of 2 objects even is 1 airline only, if not front end won't show
        expected_output = [{'carrier': 4.3562, 'nas': 0.4932, 'sec': 0.0, 'late_aircraft': 0.0, 'weather': 0.0}
        , {'carrier': 1.0153, 'nas': 0.4693, 'sec': 0.0, 'late_aircraft': 0.089, 'weather': 0.0}]

>       assert response.json() == expected_output
E       AssertionError: assert [{'carrier': ...c': 0.0, ...}] == [{'carrier': ...c': 0.0, ...}]
E         Right contains one more item: {'carrier': 1.0153, 'late_aircraft': 0.089, 'nas': 0.4693, 'sec': 0.0, ...}
E         Use -v to get more diff

test_delays_comparison.py:358: AssertionError
```

Failed *(hide details)*	test_delays_comparison.py::test_invalid_year	18.90	

```
def test_invalid_year():

        params = {'o': "SAT",
                  'dst': "PHX",
                  'a': "AA",
                  'yf' : 1985,
                  'yt' : 2030}

        with pytest.raises(ValueError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                    params=params)
E           Failed: DID NOT RAISE <class 'ValueError'>

test_delays_comparison.py:372: Failed
```

Failed *(hide details)*	test_delays_comparison.py::test_invalid_airport	20.99	

Result	Test	Duration	Links

```
def test_invalid_airport():

        params = {'o': "BBB",
                  'dst': "CCC",
                  'a': "AA",
                  'yf' : 1990,
                  'yt' : 2021}

        with pytest.raises(ValueError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                    params=params)
E           Failed: DID NOT RAISE <class 'ValueError'>

test_delays_comparison.py:385: Failed
```

Failed *(hide details)*	test_delays_comparison.py::test_invalid_flight	21.31	

```
def test_invalid_flight():

        params = {'o': "SAT",
                  'dst': "SAT",
                  'a': "AA",
                  'yf' : 1990,
                  'yt' : 2021}

        with pytest.raises(ValueError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                    params=params)
E           Failed: DID NOT RAISE <class 'ValueError'>

test_delays_comparison.py:398: Failed
```

Failed *(hide details)*	test_delays_comparison.py::test_invalid_airline	16.87	

Result	Test	Duration	Links

```
def test_invalid_airline():

        params = {'o': "SAT",
                  'dst': "PHX",
                  'a': "NOT",
                  'yf' : 1990,
                  'yt' : 2021}

        with pytest.raises(ValueError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                    params=params)
E           Failed: DID NOT RAISE <class 'ValueError'>

test_delays_comparison.py:411: Failed
```

Failed *(hide details)*	test_delays_comparison.py::test_invalid_month	0.26	

```
def test_invalid_month():

        params = {'o': "SAT",
                  'dst': "PHX",
                  'a': "AA",
                  'yf' : 1990,
                  'yt' : 2021,
                  'm' : 20}

        with pytest.raises(ValueError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                    params=params)
E           Failed: DID NOT RAISE <class 'ValueError'>

test_delays_comparison.py:425: Failed
```

Failed *(hide details)*	test_delays_comparison.py::test_invalid_day	1.30	

Result	Test	Duration	Links

```
def test_invalid_day():

        params = {'o': "SAT",
                  'dst': "PHX",
                  'a': "AA",
                  'yf' : 1990,
                  'yt' : 2021,
                  'm' : 12,
                  'd' : -1}

        with pytest.raises(ValueError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                    params=params)
E           Failed: DID NOT RAISE <class 'ValueError'>

test_delays_comparison.py:440: Failed
```

Failed *(hide details)*	test_delays_comparison.py::test_illegal_parameters	20.80	

```
def test_illegal_parameters():

        params = {'o': "SAT",
                  'dst': "PHX",
                  'a': "AA"}

        with pytest.raises(TypeError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                    params=params)
E           Failed: DID NOT RAISE <class 'TypeError'>

test_delays_comparison.py:451: Failed
```

Failed *(hide details)*	test_delays_comparison.py::test_yearFrom_greater_than_yearTo	0.23	

Result	Test	Duration	Links

```
def test_yearFrom_greater_than_yearTo():

        params = {'o': "SAT",
                  'dst': "PHX",
                  'a': "AA",
                  'yf' : 2021,
                  'yt' : 1990}

        with pytest.raises(ValueError):
>           response = requests.get("http://127.0.0.1:8080/api/flights/delays_comparison",
                    params=params)
E           Failed: DID NOT RAISE <class 'ValueError'>

test_delays_comparison.py:464: Failed
```