REPORT ON DIABETES PREDICTION

Team Members Contribution:

1. Jit Das(2206095): Drafted the initial sections of the report, including the Problem Statement and Solution Overview.
2. Debdip Chatterjee(2206087): Provided additional proofreading and quality checks to enhance the overall presentation of the report.
3. Ritesh Chowdhury(2206369): Collaborated on project development with Debdip, focusing on implementing the Logistic Regression model for binary classification.
4. Shreya Saha(2206379):Reviewed and edited all sections for coherence, clarity, and technical accuracy.
5. Aryan Yadav(2206168):Collaborated on model development, with a focus on coding the main logistic regression functions.
6. Soumya Bhattacharjee(2206219): Organized the final report structure, ensuring logical flow and adherence to formatting guidelines.

1. Problem Statement:

The problem statement for the document "REPORT ON DIABETES PREDICTION" focuses on predicting diabetes using a machine learning model. The report includes a data set with various health metrics, such as pregnancies, glucose levels, blood pressure, skin thickness, insulin levels, BMI, diabetes pedigree function, and age. The goal is to classify individuals as either diabetic (1) or non-diabetic (0) using logistic regression for binary classification. This involves analyzing these metrics to develop a predictive model that can assist in early detection of diabetes.

2. Brief Overview of Solution:

The solution to the diabetes prediction problem involves using a logistic regression model for binary classification. Here's a brief overview of the approach taken:

1. Data Analysis and Prepossessing: The data set is analyzed to understand the distribution of each feature, identify missing or outlier values, and perform any necessary cleaning. This step ensures that the model has high-quality input data.

2. Feature Selection and Scaling: Relevant features from the data set, such as glucose levels, BMI, and insulin, are selected for training. Features are also scaled to improve model accuracy and convergence.

3. Model Selection (Logistic Regression): Logistic regression is chosen for its simplicity and effectiveness in binary classification tasks. This model is well-suited for predicting probabilities, making it useful in identifying the likelihood of diabetes.

4. Model Training and Evaluation: The logistic regression model is trained using the processed data set. It is then evaluated on a test set to measure accuracy, precision, recall, and other metrics to ensure reliable prediction performance.

5. Outcome Prediction: After validation, the model can predict diabetes outcomes for new input data, helping in early identification and potentially guiding preventive healthcare decisions.

This solution emphasizes accuracy, interpretability, and practical applicability for predicting diabetes in a healthcare setting.

3. **Description of Dataset:**

The data set has the following columns:

Pregnancies: Number of pregnancies the person has had.
Glucose: Plasma glucose concentration after a 2-hour oral glucose tolerance test.
Blood Pressure: Diastolic blood pressure (mm Hg).
Skin Thickness: Skin fold thickness (mm).
Insulin: 2-hour serum insulin (mu U/ml).
BMI: Body mass index (weight in kg / (height in m)^2).
Diabetes Pedigree Function: A function that scores the likelihood of diabetes based on family history.
Age: Age of the person (in years).
Outcome: The target variable (0 = Non-Diabetic, 1 = Diabetic).

diabetes

November 7, 2024

Importing the Dependencies

[1]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
```

Data Collection and Analysis

PIMA Diabetes Dataset

[2]:
```
# loading the diabetes dataset to a pandas DataFrame
diabetes_dataset = pd.read_csv("C:\\Users\\KIIT\\Downloads\\diabetes.csv")
```

[3]:
```
pd.read_csv?
```

```
Signature:
pd.read_csv(
    filepath_or_buffer: 'FilePath |
ReadCsvBuffer[bytes] | ReadCsvBuffer[str]',
    *,
    sep: 'str | None | lib.NoDefault'
= <no_default>,
    delimiter: 'str | None | lib.NoDefault'
= None,
    header: "int | Sequence[int] | None |
Literal['infer']" = 'infer',
    names: 'Sequence[Hashable] | None |
lib.NoDefault' =
```

```
<no_default>,
    index_col: 'IndexLabel | Literal[False] |
None' = None,
    usecols: 'UsecolsArgType' =
None,
    dtype: 'DtypeArg | None' =
None,
    engine: 'CSVEngine | None' =
None,
    converters: 'Mapping[Hashable, Callable] |
None' = None,
    true_values: 'list | None' =
None,
    false_values: 'list | None' =
None,
    skipinitialspace: 'bool' =
False,
    skiprows: 'list[int] | int |
Callable[[Hashable], bool] | None' =
None,
    skipfooter: 'int' =
0,
    nrows: 'int | None' =
None,
    na_values: 'Hashable | Iterable[Hashable] |
Mapping[Hashable, Iterable[Hashable]] | None' =
None,
    keep_default_na: 'bool' =
True,
    na_filter: 'bool' =
True,
    verbose: 'bool | lib.NoDefault'
= <no_default>,
    skip_blank_lines: 'bool' =
```

```
True,
    parse_dates: 'bool | Sequence[Hashable] |
None' = None,
    infer_datetime_format: 'bool |
lib.NoDefault' =
<no_default>,
    keep_date_col: 'bool | lib.NoDefault'
= <no_default>,
    date_parser: 'Callable | lib.NoDefault'
= <no_default>,
    date_format: 'str | dict[Hashable, str] |
None' = None,
    dayfirst: 'bool' =
False,
    cache_dates: 'bool' =
True,
    iterator: 'bool' =
False,
    chunksize: 'int | None' =
None,
    compression: 'CompressionOptions'
= 'infer',
    thousands: 'str | None' =
None,
    decimal: 'str' =
'.',
    lineterminator: 'str | None' =
None,
    quotechar: 'str' =
'"',
    quoting: 'int' =
0,
    doublequote: 'bool' =
True,
    escapechar: 'str | None' =
```

```
None,
    comment: 'str | None' =
None,
    encoding: 'str | None' =
None,
    encoding_errors: 'str | None' =
'strict',
    dialect: 'str | csv.Dialect | None'
= None,
    on_bad_lines: 'str' =
'error',
    delim_whitespace: 'bool | lib.NoDefault'
= <no_default>,
    low_memory: 'bool' =
True,
    memory_map: 'bool' =
False,
    float_precision: "Literal['high', 'legacy'] |
None" = None,
    storage_options: 'StorageOptions | None'
= None,
    dtype_backend: 'DtypeBackend |
lib.NoDefault' =
<no_default>,
) -> 'DataFrame |
TextFileReader'
Docstring:
Read a comma-separated values (csv) file into DataFrame.

Also supports optionally iterating or breaking of the file
into chunks.

Additional help can be found in the online docs for
`IO Tools <https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html>`_.

Parameters
----------
filepath_or_buffer : str, path object or file-like object
    Any valid string path is acceptable. The string could be a URL. Valid
```

```
    URL schemes include http, ftp, s3, gs, and file. For file URLs, a host is
    expected. A local file could be: file://localhost/path/to/table.csv.

    If you want to pass in a path object, pandas accepts any ``os.PathLike``.

    By file-like object, we refer to objects with a ``read()`` method, such as
    a file handle (e.g. via builtin ``open`` function) or ``StringIO``.
sep : str, default ','
    Character or regex pattern to treat as the delimiter. If ``sep=None``, the
    C engine cannot automatically detect
    the separator, but the Python parsing engine can, meaning the latter will
    be used and automatically detect the separator from only the first valid
    row of the file by Python's builtin sniffer tool, ``csv.Sniffer``.
    In addition, separators longer than 1 character and different from
    ``'\s+'`` will be interpreted as regular expressions and will also force
    the use of the Python parsing engine. Note that regex delimiters are prone
    to ignoring quoted data. Regex example: ``'\r\t'``.
delimiter : str, optional
    Alias for ``sep``.
header : int, Sequence of int, 'infer' or None, default 'infer'
    Row number(s) containing column labels and marking the start of the
    data (zero-indexed). Default behavior is to infer the column names: if no
``names``
    are passed the behavior is identical to ``header=0`` and column
    names are inferred from the first line of the file, if column
    names are passed explicitly to ``names`` then the behavior is identical to
    ``header=None``. Explicitly pass ``header=0`` to be able to
    replace existing names. The header can be a list of integers that
    specify row locations for a :class:`~pandas.MultiIndex` on the columns
    e.g. ``[0, 1, 3]``. Intervening rows that are not specified will be
    skipped (e.g. 2 in this example is skipped). Note that this
    parameter ignores commented lines and empty lines if
    ``skip_blank_lines=True``, so ``header=0`` denotes the first line of
    data rather than the first line of the file.
names : Sequence of Hashable, optional
    Sequence of column labels to apply. If the file contains a header row,
    then you should explicitly pass ``header=0`` to override the column names.
    Duplicates in this list are not allowed.
index_col : Hashable, Sequence of Hashable or False, optional
  Column(s) to use as row label(s), denoted either by column labels or column
  indices.  If a sequence of labels or indices is given,
:class:`~pandas.MultiIndex`
  will be formed for the row labels.

  Note: ``index_col=False`` can be used to force pandas to *not* use the first
  column as the index, e.g., when you have a malformed file with delimiters at
  the end of each line.
usecols : Sequence of Hashable or Callable, optional
```

```
    Subset of columns to select, denoted either by column labels or column
indices.
    If list-like, all elements must either
    be positional (i.e. integer indices into the document columns) or strings
    that correspond to column names provided either by the user in ``names`` or
    inferred from the document header row(s). If ``names`` are given, the
document
    header row(s) are not taken into account. For example, a valid list-like
    ``usecols`` parameter would be ``[0, 1, 2]`` or ``['foo', 'bar', 'baz']``.
    Element order is ignored, so ``usecols=[0, 1]`` is the same as ``[1, 0]``.
    To instantiate a :class:`~pandas.DataFrame` from ``data`` with element order
    preserved use ``pd.read_csv(data, usecols=['foo', 'bar'])[['foo', 'bar']]``
    for columns in ``['foo', 'bar']`` order or
    ``pd.read_csv(data, usecols=['foo', 'bar'])[['bar', 'foo']]``
    for ``['bar', 'foo']`` order.

    If callable, the callable function will be evaluated against the column
    names, returning names where the callable function evaluates to ``True``. An
    example of a valid callable argument would be ``lambda x: x.upper() in
    ['AAA', 'BBB', 'DDD']``. Using this parameter results in much faster
    parsing time and lower memory usage.
dtype : dtype or dict of {Hashable : dtype}, optional
    Data type(s) to apply to either the whole dataset or individual columns.
    E.g., ``{'a': np.float64, 'b': np.int32, 'c': 'Int64'}``
    Use ``str`` or ``object`` together with suitable ``na_values`` settings
    to preserve and not interpret ``dtype``.
    If ``converters`` are specified, they will be applied INSTEAD
    of ``dtype`` conversion.

    .. versionadded:: 1.5.0

        Support for ``defaultdict`` was added. Specify a ``defaultdict`` as
input where
        the default determines the ``dtype`` of the columns which are not
explicitly
        listed.
engine : {'c', 'python', 'pyarrow'}, optional
    Parser engine to use. The C and pyarrow engines are faster, while the python
engine
    is currently more feature-complete. Multithreading is currently only
supported by
    the pyarrow engine.

    .. versionadded:: 1.4.0

        The 'pyarrow' engine was added as an *experimental* engine, and some
features
        are unsupported, or may not work correctly, with this engine.
```

```
converters : dict of {Hashable : Callable}, optional
    Functions for converting values in specified columns. Keys can either
    be column labels or column indices.
true_values : list, optional
    Values to consider as ``True`` in addition to case-insensitive variants of
'True'.
false_values : list, optional
    Values to consider as ``False`` in addition to case-insensitive variants of
'False'.
skipinitialspace : bool, default False
    Skip spaces after delimiter.
skiprows : int, list of int or Callable, optional
    Line numbers to skip (0-indexed) or number of lines to skip (``int``)
    at the start of the file.

    If callable, the callable function will be evaluated against the row
    indices, returning ``True`` if the row should be skipped and ``False``
otherwise.
    An example of a valid callable argument would be ``lambda x: x in [0, 2]``.
skipfooter : int, default 0
    Number of lines at bottom of file to skip (Unsupported with ``engine='c'``).
nrows : int, optional
    Number of rows of file to read. Useful for reading pieces of large files.
na_values : Hashable, Iterable of Hashable or dict of {Hashable : Iterable},
optional
    Additional strings to recognize as ``NA``/``NaN``. If ``dict`` passed,
specific
    per-column ``NA`` values.  By default the following values are interpreted
as
    ``NaN``: " ", "#N/A", "#N/A N/A", "#NA", "-1.#IND", "-1.#QNAN", "-NaN",
"-nan",
    "1.#IND", "1.#QNAN", "<NA>", "N/A", "NA", "NULL", "NaN", "None",
    "n/a", "nan", "null ".

keep_default_na : bool, default True
    Whether or not to include the default ``NaN`` values when parsing the data.
    Depending on whether ``na_values`` is passed in, the behavior is as follows:

    * If ``keep_default_na`` is ``True``, and ``na_values`` are specified,
``na_values``
      is appended to the default ``NaN`` values used for parsing.
    * If ``keep_default_na`` is ``True``, and ``na_values`` are not specified,
only
      the default ``NaN`` values are used for parsing.
    * If ``keep_default_na`` is ``False``, and ``na_values`` are specified, only
      the ``NaN`` values specified ``na_values`` are used for parsing.
    * If ``keep_default_na`` is ``False``, and ``na_values`` are not specified,
no
```

```
      strings will be parsed as ``NaN``.

    Note that if ``na_filter`` is passed in as ``False``, the
``keep_default_na`` and
    ``na_values`` parameters will be ignored.
na_filter : bool, default True
    Detect missing value markers (empty strings and the value of ``na_values``).
In
    data without any ``NA`` values, passing ``na_filter=False`` can improve the
    performance of reading a large file.
verbose : bool, default False
    Indicate number of ``NA`` values placed in non-numeric columns.

    .. deprecated:: 2.2.0
skip_blank_lines : bool, default True
    If ``True``, skip over blank lines rather than interpreting as ``NaN``
values.
parse_dates : bool, list of Hashable, list of lists or dict of {Hashable :
list}, default False
    The behavior is as follows:

    * ``bool``. If ``True`` -> try parsing the index. Note: Automatically set to
      ``True`` if ``date_format`` or ``date_parser`` arguments have been passed.
    * ``list`` of ``int`` or names. e.g. If ``[1, 2, 3]`` -> try parsing columns
1, 2, 3
      each as a separate date column.
    * ``list`` of ``list``. e.g.  If ``[[1, 3]]`` -> combine columns 1 and 3 and
parse
      as a single date column. Values are joined with a space before parsing.
    * ``dict``, e.g. ``{'foo' : [1, 3]}`` -> parse columns 1, 3 as date and call
      result 'foo'. Values are joined with a space before parsing.

    If a column or index cannot be represented as an array of ``datetime``,
    say because of an unparsable value or a mixture of timezones, the column
    or index will be returned unaltered as an ``object`` data type. For
    non-standard ``datetime`` parsing, use :func:`~pandas.to_datetime` after
    :func:`~pandas.read_csv`.

    Note: A fast-path exists for iso8601-formatted dates.
infer_datetime_format : bool, default False
    If ``True`` and ``parse_dates`` is enabled, pandas will attempt to infer the
    format of the ``datetime`` strings in the columns, and if it can be
inferred,
    switch to a faster method of parsing them. In some cases this can increase
    the parsing speed by 5-10x.

    .. deprecated:: 2.0.0
        A strict version of this argument is now the default, passing it has no
```

```
effect.

keep_date_col : bool, default False
    If ``True`` and ``parse_dates`` specifies combining multiple columns then
    keep the original columns.
date_parser : Callable, optional
    Function to use for converting a sequence of string columns to an array of
    ``datetime`` instances. The default uses ``dateutil.parser.parser`` to do
the
    conversion. pandas will try to call ``date_parser`` in three different ways,
    advancing to the next if an exception occurs: 1) Pass one or more arrays
    (as defined by ``parse_dates``) as arguments; 2) concatenate (row-wise) the
    string values from the columns defined by ``parse_dates`` into a single
array
    and pass that; and 3) call ``date_parser`` once for each row using one or
    more strings (corresponding to the columns defined by ``parse_dates``) as
    arguments.

    .. deprecated:: 2.0.0
       Use ``date_format`` instead, or read in as ``object`` and then apply
       :func:`~pandas.to_datetime` as-needed.
date_format : str or dict of column -> format, optional
    Format to use for parsing dates when used in conjunction with
``parse_dates``.
    The strftime to parse time, e.g. :const:`"%d/%m/%Y"`. See
    `strftime documentation
    <https://docs.python.org/3/library/datetime.html
    #strftime-and-strptime-behavior>`_ for more information on choices, though
    note that :const:`"%f"` will parse all the way up to nanoseconds.
    You can also pass:

    - "ISO8601", to parse any `ISO8601
<https://en.wikipedia.org/wiki/ISO_8601>`_
        time string (not necessarily in exactly the same format);
    - "mixed", to infer the format for each element individually. This is risky,
        and you should probably use it along with `dayfirst`.

    .. versionadded:: 2.0.0
dayfirst : bool, default False
    DD/MM format dates, international and European format.
cache_dates : bool, default True
    If ``True``, use a cache of unique, converted dates to apply the
``datetime``
    conversion. May produce significant speed-up when parsing duplicate
    date strings, especially ones with timezone offsets.

iterator : bool, default False
    Return ``TextFileReader`` object for iteration or getting chunks with
```

```
    ``get_chunk()``.
chunksize : int, optional
    Number of lines to read from the file per chunk. Passing a value will cause
the
    function to return a ``TextFileReader`` object for iteration.
    See the `IO Tools docs
    <https://pandas.pydata.org/pandas-docs/stable/io.html#io-chunking>`_
    for more information on ``iterator`` and ``chunksize``.

compression : str or dict, default 'infer'
    For on-the-fly decompression of on-disk data. If 'infer' and
'filepath_or_buffer' is
    path-like, then detect compression from the following extensions: '.gz',
    '.bz2', '.zip', '.xz', '.zst', '.tar', '.tar.gz', '.tar.xz' or '.tar.bz2'
    (otherwise no compression).
    If using 'zip' or 'tar', the ZIP file must contain only one data file to be
read in.
    Set to ``None`` for no decompression.
    Can also be a dict with key ``'method'`` set
    to one of {``'zip'``, ``'gzip'``, ``'bz2'``, ``'zstd'``, ``'xz'``,
``'tar'``} and
    other key-value pairs are forwarded to
    ``zipfile.ZipFile``, ``gzip.GzipFile``,
    ``bz2.BZ2File``, ``zstandard.ZstdDecompressor``, ``lzma.LZMAFile`` or
    ``tarfile.TarFile``, respectively.
    As an example, the following could be passed for Zstandard decompression
using a
    custom compression dictionary:
    ``compression={'method': 'zstd', 'dict_data': my_compression_dict}``.

    .. versionadded:: 1.5.0
        Added support for `.tar` files.

    .. versionchanged:: 1.4.0 Zstandard support.

thousands : str (length 1), optional
    Character acting as the thousands separator in numerical values.
decimal : str (length 1), default '.'
    Character to recognize as decimal point (e.g., use ',' for European data).
lineterminator : str (length 1), optional
    Character used to denote a line break. Only valid with C parser.
quotechar : str (length 1), optional
    Character used to denote the start and end of a quoted item. Quoted
    items can include the ``delimiter`` and it will be ignored.
quoting : {0 or csv.QUOTE_MINIMAL, 1 or csv.QUOTE_ALL, 2 or
csv.QUOTE_NONNUMERIC, 3 or csv.QUOTE_NONE}, default csv.QUOTE_MINIMAL
    Control field quoting behavior per ``csv.QUOTE_*`` constants. Default is
    ``csv.QUOTE_MINIMAL`` (i.e., 0) which implies that only fields containing
```

```
special
    characters are quoted (e.g., characters defined in ``quotechar``,
``delimiter``,
    or ``lineterminator``.
doublequote : bool, default True
   When ``quotechar`` is specified and ``quoting`` is not ``QUOTE_NONE``,
indicate
   whether or not to interpret two consecutive ``quotechar`` elements INSIDE a
   field as a single ``quotechar`` element.
escapechar : str (length 1), optional
    Character used to escape other characters.
comment : str (length 1), optional
    Character indicating that the remainder of line should not be parsed.
    If found at the beginning
    of a line, the line will be ignored altogether. This parameter must be a
    single character. Like empty lines (as long as ``skip_blank_lines=True``),
    fully commented lines are ignored by the parameter ``header`` but not by
    ``skiprows``. For example, if ``comment='#'``, parsing
    ``#empty\na,b,c\n1,2,3`` with ``header=0`` will result in ``'a,b,c'`` being
    treated as the header.
encoding : str, optional, default 'utf-8'
    Encoding to use for UTF when reading/writing (ex. ``'utf-8'``). `List of
Python
    standard encodings
    <https://docs.python.org/3/library/codecs.html#standard-encodings>`_ .

encoding_errors : str, optional, default 'strict'
    How encoding errors are treated. `List of possible values
    <https://docs.python.org/3/library/codecs.html#error-handlers>`_ .

    .. versionadded:: 1.3.0

dialect : str or csv.Dialect, optional
    If provided, this parameter will override values (default or not) for the
    following parameters: ``delimiter``, ``doublequote``, ``escapechar``,
    ``skipinitialspace``, ``quotechar``, and ``quoting``. If it is necessary to
    override values, a ``ParserWarning`` will be issued. See ``csv.Dialect``
    documentation for more details.
on_bad_lines : {'error', 'warn', 'skip'} or Callable, default 'error'
    Specifies what to do upon encountering a bad line (a line with too many
fields).
    Allowed values are :

    - ``'error'``, raise an Exception when a bad line is encountered.
    - ``'warn'``, raise a warning when a bad line is encountered and skip that
line.
    - ``'skip'``, skip bad lines without raising or warning when they are
encountered.
```

```
    .. versionadded:: 1.3.0

    .. versionadded:: 1.4.0

        - Callable, function with signature
          ``(bad_line: list[str]) -> list[str] | None`` that will process a
single
          bad line. ``bad_line`` is a list of strings split by the ``sep``.
          If the function returns ``None``, the bad line will be ignored.
          If the function returns a new ``list`` of strings with more elements
than
          expected, a ``ParserWarning`` will be emitted while dropping extra
elements.
          Only supported when ``engine='python'``

    .. versionchanged:: 2.2.0

        - Callable, function with signature
          as described in `pyarrow documentation
<https://arrow.apache.org/docs/python/generated/pyarrow.csv.ParseOptions.html
          #pyarrow.csv.ParseOptions.invalid_row_handler>`_ when
``engine='pyarrow'``

delim_whitespace : bool, default False
    Specifies whether or not whitespace (e.g. ``' '`` or ``'\t'``) will be
    used as the ``sep`` delimiter. Equivalent to setting ``sep='\s+'``. If this
option
    is set to ``True``, nothing should be passed in for the ``delimiter``
    parameter.

    .. deprecated:: 2.2.0
        Use ``sep="\s+"`` instead.
low_memory : bool, default True
    Internally process the file in chunks, resulting in lower memory use
    while parsing, but possibly mixed type inference.  To ensure no mixed
    types either set ``False``, or specify the type with the ``dtype``
parameter.
    Note that the entire file is read into a single :class:`~pandas.DataFrame`
    regardless, use the ``chunksize`` or ``iterator`` parameter to return the
data in
    chunks. (Only valid with C parser).
memory_map : bool, default False
    If a filepath is provided for ``filepath_or_buffer``, map the file object
    directly onto memory and access the data directly from there. Using this
    option can improve performance because there is no longer any I/O overhead.
float_precision : {'high', 'legacy', 'round_trip'}, optional
    Specifies which converter the C engine should use for floating-point
```

```
    values. The options are ``None`` or ``'high'`` for the ordinary converter,
    ``'legacy'`` for the original lower precision pandas converter, and
    ``'round_trip'`` for the round-trip converter.

storage_options : dict, optional
    Extra options that make sense for a particular storage connection, e.g.
    host, port, username, password, etc. For HTTP(S) URLs the key-value pairs
    are forwarded to ``urllib.request.Request`` as header options. For other
    URLs (e.g. starting with "s3://", and "gcs://") the key-value pairs are
    forwarded to ``fsspec.open``. Please see ``fsspec`` and ``urllib`` for more
    details, and for more examples on storage options refer `here
    <https://pandas.pydata.org/docs/user_guide/io.html?
    highlight=storage_options#reading-writing-remote-files>`_.

dtype_backend : {'numpy_nullable', 'pyarrow'}, default 'numpy_nullable'
    Back-end data type applied to the resultant :class:`DataFrame`
    (still experimental). Behaviour is as follows:

    * ``"numpy_nullable"``: returns nullable-dtype-backed :class:`DataFrame`
      (default).
    * ``"pyarrow"``: returns pyarrow-backed nullable :class:`ArrowDtype`
      DataFrame.

    .. versionadded:: 2.0

Returns
-------
DataFrame or TextFileReader
    A comma-separated values (csv) file is returned as two-dimensional
    data structure with labeled axes.

See Also
--------
DataFrame.to_csv : Write DataFrame to a comma-separated values (csv) file.
read_table : Read general delimited file into DataFrame.
read_fwf : Read a table of fixed-width formatted lines into DataFrame.

Examples
--------
>>> pd.read_csv('data.csv')  # doctest: +SKIP
File:
c:\users\kiit\appdata\local\programs\python\python310\lib\site-
packages\pandas\io\parsers\readers.py
Type:      function
```

[4]:
```
# printing the first 5 rows of the dataset
diabetes_dataset.head()
```

[4]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[5]:
```
# number of rows and Columns in this dataset
diabetes_dataset.shape
```

[5]:
```
(768, 9)
```

[6]:
```
# getting the statistical measures of the data
diabetes_dataset.describe()
```

[6]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

[7]:
```
diabetes_dataset['Outcome'].value_counts()
```

[7]:
```
Outcome
0    500
1    268
```

```
Name: count, dtype: int64
```

0 –> Non-Diabetic

1 –> Diabetic

[8]:
```
diabetes_dataset.groupby('Outcome').mean()
```

[8]:
```
         Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
Outcome
0           3.298000  109.980000      68.184000      19.664000   68.792000
1           4.865672  141.257463      70.824627      22.164179  100.335821

                BMI  DiabetesPedigreeFunction        Age
Outcome
0         30.304200                  0.429734  31.190000
1         35.142537                  0.550500  37.067164
```

[9]:
```
# separating the data and labels
X = diabetes_dataset.drop(columns = 'Outcome', axis=1)
Y = diabetes_dataset['Outcome']
```

[10]:
```
print(X)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           ...      ...            ...            ...      ...   ...
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age
0                       0.627   50
1                       0.351   31
2                       0.672   32
3                       0.167   21
4                       2.288   33
..                        ...  ...
763                     0.171   63
764                     0.340   27
765                     0.245   30
766                     0.349   47
```

```
767                    0.315   23

[768 rows x 8 columns]
```

[11]:
```
print(Y)
```

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

Data Standardization

[12]:
```
scaler = StandardScaler()
```

[13]:
```
scaler.fit(X)
```

[13]:
```
StandardScaler()
```

[14]:
```
standardized_data = scaler.transform(X)
```

[15]:
```
print(standardized_data)
```

```
[[ 0.63994726  0.84832379  0.14964075 …  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 … -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 … -1.10325546  0.60439732
  -0.10558415]
 …
 [ 0.3429808   0.00330087  0.14964075 … -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 … -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 … -0.20212881 -0.47378505
  -0.87137393]]
```

[16]:
```
X = standardized_data
Y = diabetes_dataset['Outcome']
```

[17]:
```
print(X)
print(Y)
```

```
[[ 0.63994726  0.84832379  0.14964075 …  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 … -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 … -1.10325546  0.60439732
  -0.10558415]
 …
 [ 0.3429808   0.00330087  0.14964075 … -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 … -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 … -0.20212881 -0.47378505
  -0.87137393]]
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

Train Test Split

[18]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2,␣
↪stratify=Y, random_state=2)
```

[19]:
```
print(X.shape, X_train.shape, X_test.shape)
```

```
(768, 8) (614, 8) (154, 8)
```

Training the Model

[20]:
```
classifier = svm.SVC(kernel='linear')
```

[21]:
```
#training the support vector Machine Classifier
classifier.fit(X_train, Y_train)
```

[21]:
```
SVC(kernel='linear')
```

Model Evaluation

Accuracy Score

[22]:
```
# accuracy score on the training data
X_train_prediction = classifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

[23]:
```
print('Accuracy score of the training data : ', training_data_accuracy)
```

```
Accuracy score of the training data :  0.7866449511400652
```

[24]:
```
# accuracy score on the test data
X_test_prediction = classifier.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

[25]:
```
print('Accuracy score of the test data : ', test_data_accuracy)
```

```
Accuracy score of the test data :  0.7727272727272727
```

Making a Predictive System

[26]:
```
input_data = (5,166,72,19,175,25.8,0.587,51)

# changing the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardize the input data
std_data = scaler.transform(input_data_reshaped)
print(std_data)

prediction = classifier.predict(std_data)
print(prediction)

if (prediction[0] == 0):
  print('The person is not diabetic')
else:
  print('The person is diabetic')
```

```
[[ 0.3429808   1.41167241  0.14964075 -0.09637905  0.82661621 -0.78595734
   0.34768723  1.51108316]]
[1]
The person is diabetic

c:\Users\KIIT\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\base.py:493: UserWarning: X does not have valid feature names,
but StandardScaler was fitted with feature names
  warnings.warn(
```

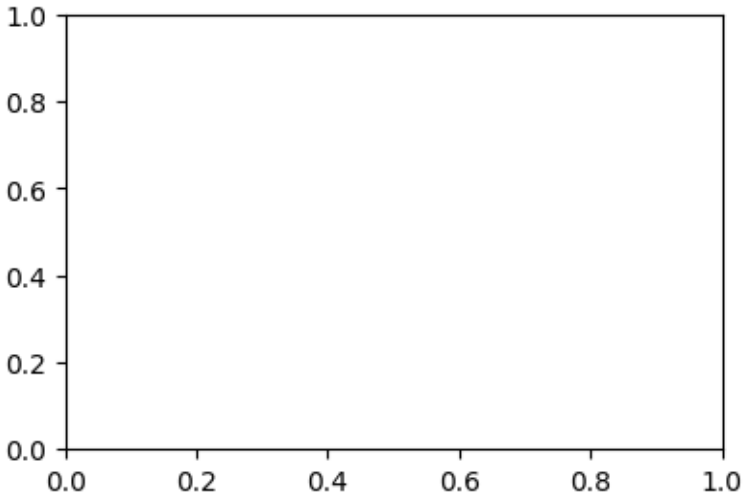
[27]:
```
!pip install seaborn
```

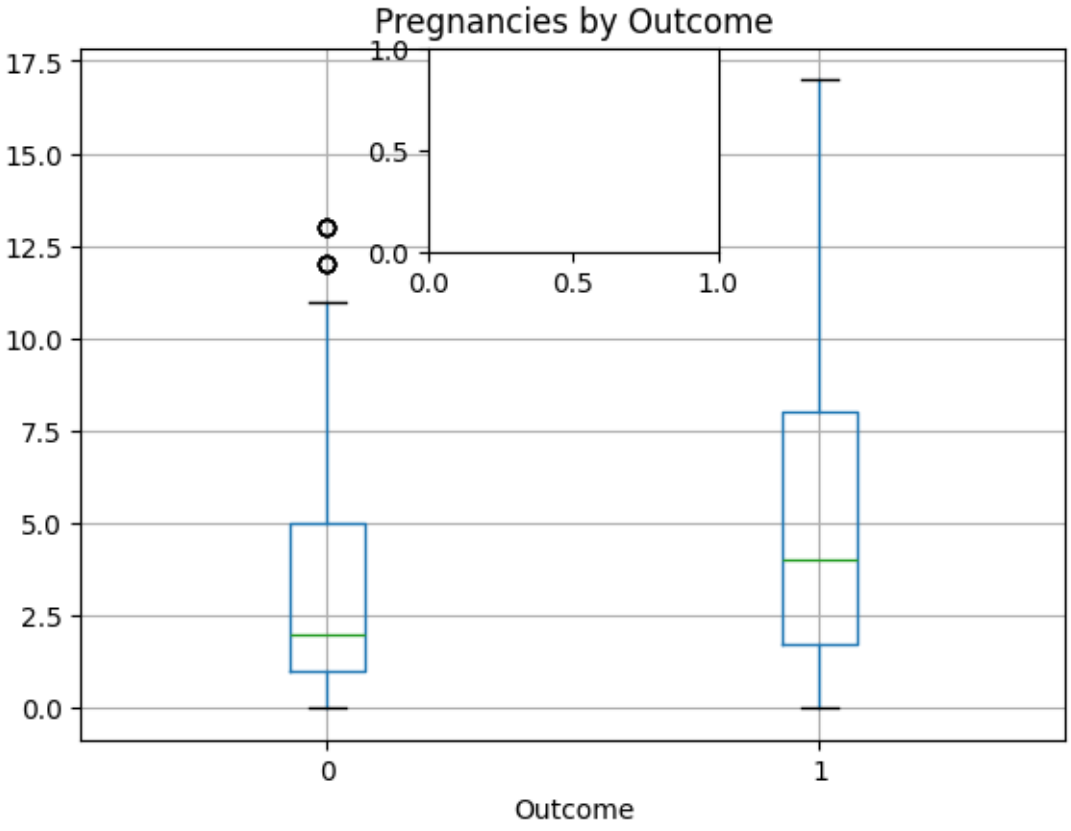
```
Requirement already satisfied: seaborn in
c:\users\kiit\appdata\local\programs\python\python310\lib\site-packages (0.13.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in
c:\users\kiit\appdata\local\programs\python\python310\lib\site-packages (from
seaborn) (3.9.2)
Requirement already satisfied: pandas>=1.2 in
c:\users\kiit\appdata\local\programs\python\python310\lib\site-packages (from
seaborn) (2.2.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in
c:\users\kiit\appdata\local\programs\python\python310\lib\site-packages (from
seaborn) (2.0.0)
Requirement already satisfied: contourpy>=1.0.1 in
c:\users\kiit\appdata\local\programs\python\python310\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.3.0)
Requirement already satisfied: cycler>=0.10 in
c:\users\kiit\appdata\local\programs\python\python310\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: pyparsing>=2.3.1 in
c:\users\kiit\appdata\roaming\python\python310\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (3.0.9)
Requirement already satisfied: pillow>=8 in
c:\users\kiit\appdata\local\programs\python\python310\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (11.0.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
c:\users\kiit\appdata\local\programs\python\python310\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.4.7)
Requirement already satisfied: packaging>=20.0 in
c:\users\kiit\appdata\roaming\python\python310\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (21.3)
Requirement already satisfied: python-dateutil>=2.7 in
c:\users\kiit\appdata\roaming\python\python310\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (2.8.2)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\kiit\appdata\local\programs\python\python310\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (4.54.1)
Requirement already satisfied: pytz>=2020.1 in
c:\users\kiit\appdata\local\programs\python\python310\lib\site-packages (from
pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in
c:\users\kiit\appdata\local\programs\python\python310\lib\site-packages (from
pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: six>=1.5 in
c:\users\kiit\appdata\roaming\python\python310\site-packages (from python-
dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)

[notice] A new release of pip available: 22.2.2 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

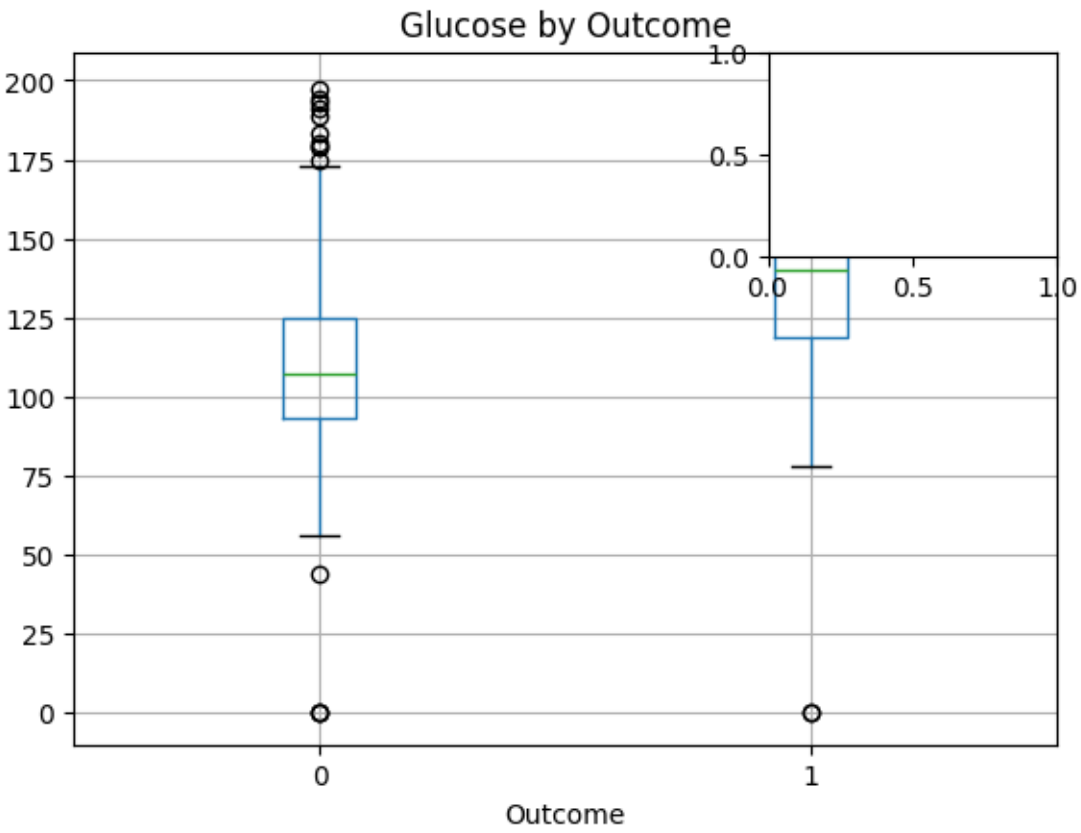
[28]:
```
# Import required libraries
import matplotlib.pyplot as plt
import numpy as np

# 5. Box plots for all features
plt.figure(figsize=(15, 10))
for i, column in enumerate(diabetes_dataset.columns[:-1], 1):
    plt.subplot(3, 3, i)
    diabetes_dataset.boxplot(column=column, by='Outcome')
    plt.title(f'{column} by Outcome')
    plt.suptitle('')  # This removes the automatic suptitle
plt.tight_layout()
plt.show()

# 6. Density plots for each feature
plt.figure(figsize=(15, 10))
for i, column in enumerate(diabetes_dataset.columns[:-1], 1):
    plt.subplot(3, 3, i)
    for outcome in [0, 1]:
        data = diabetes_dataset[diabetes_dataset['Outcome'] == outcome][column]
        plt.hist(data, density=True, alpha=0.5, label=f'Outcome {outcome}',␣
 ↪bins=30)
    plt.title(f'{column} Distribution')
    plt.legend()
plt.tight_layout()
plt.show()

# 7. Age distribution analysis
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.hist(diabetes_dataset[diabetes_dataset['Outcome']==0]['Age'],
         alpha=0.5, label='No Diabetes', bins=20)
plt.hist(diabetes_dataset[diabetes_dataset['Outcome']==1]['Age'],
         alpha=0.5, label='Diabetes', bins=20)
plt.title('Age Distribution by Outcome')
plt.legend()

plt.subplot(1, 2, 2)
age_groups = pd.cut(diabetes_dataset['Age'], bins=[0, 20, 30, 40, 50, 60, 100])
diabetes_dataset.groupby([age_groups, 'Outcome']).size().unstack().
 ↪plot(kind='bar')
plt.title('Diabetes Cases by Age Group')
plt.tight_layout()
plt.show()

# 8. Multiple feature relationships
plt.figure(figsize=(15, 5))
```

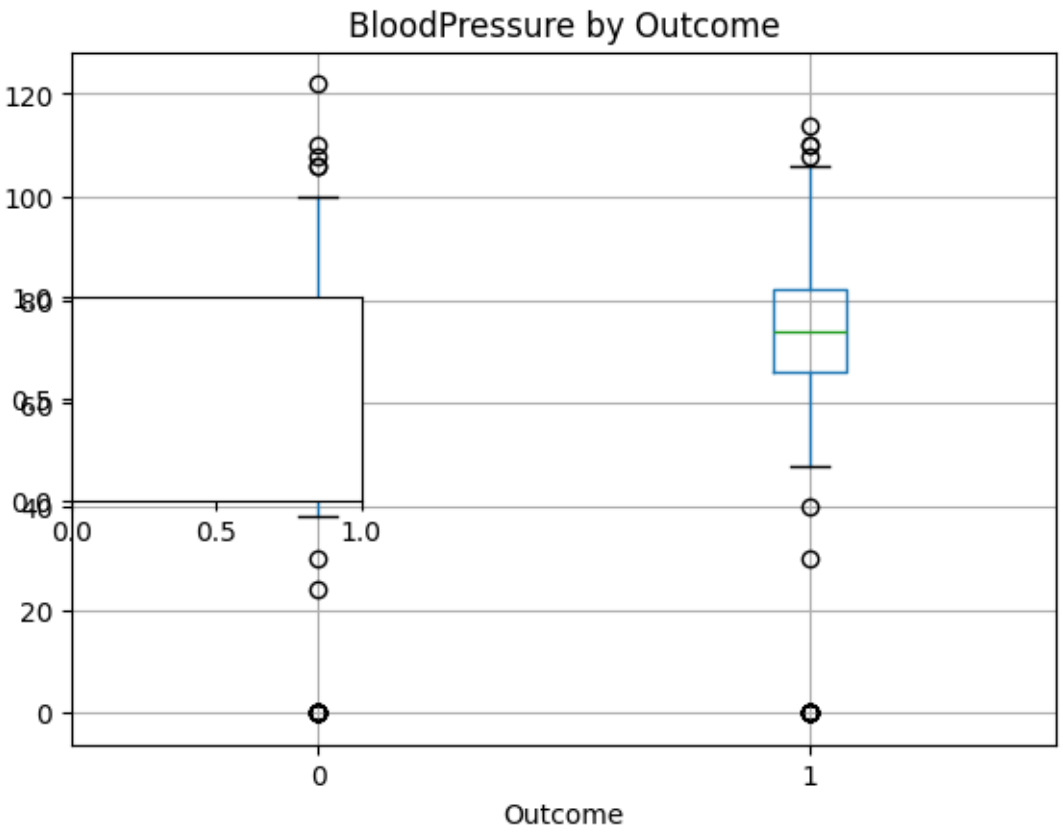
```
# Glucose vs BMI
plt.subplot(1, 3, 1)
plt.scatter(diabetes_dataset['Glucose'], diabetes_dataset['BMI'],
            c=diabetes_dataset['Age'], cmap='viridis')
plt.colorbar(label='Age')
plt.xlabel('Glucose')
plt.ylabel('BMI')
plt.title('Glucose vs BMI (colored by Age)')

# Glucose vs Blood Pressure
plt.subplot(1, 3, 2)
plt.scatter(diabetes_dataset['Glucose'], diabetes_dataset['BloodPressure'],
            c=diabetes_dataset['Age'], cmap='viridis')
plt.colorbar(label='Age')
plt.xlabel('Glucose')
plt.ylabel('Blood Pressure')
plt.title('Glucose vs Blood Pressure (colored by Age)')

# BMI vs Blood Pressure
plt.subplot(1, 3, 3)
plt.scatter(diabetes_dataset['BMI'], diabetes_dataset['BloodPressure'],
            c=diabetes_dataset['Age'], cmap='viridis')
plt.colorbar(label='Age')
plt.xlabel('BMI')
plt.ylabel('Blood Pressure')
plt.title('BMI vs Blood Pressure (colored by Age)')
plt.tight_layout()
plt.show()

# 9. Feature distributions with mean lines
plt.figure(figsize=(15, 10))
for i, column in enumerate(diabetes_dataset.columns[:-1], 1):
    plt.subplot(3, 3, i)
    plt.hist(diabetes_dataset[column], bins=30, alpha=0.5)
    plt.axvline(diabetes_dataset[column].mean(), color='red',␣
 ↪linestyle='dashed', linewidth=1)
    plt.axvline(diabetes_dataset[column].median(), color='green',␣
 ↪linestyle='dashed', linewidth=1)
    plt.title(f'{column} Distribution\nRed: Mean, Green: Median')
plt.tight_layout()
plt.show()

# 10. Cumulative distribution plots
plt.figure(figsize=(15, 10))
for i, column in enumerate(diabetes_dataset.columns[:-1], 1):
    plt.subplot(3, 3, i)
    for outcome in [0, 1]:
```

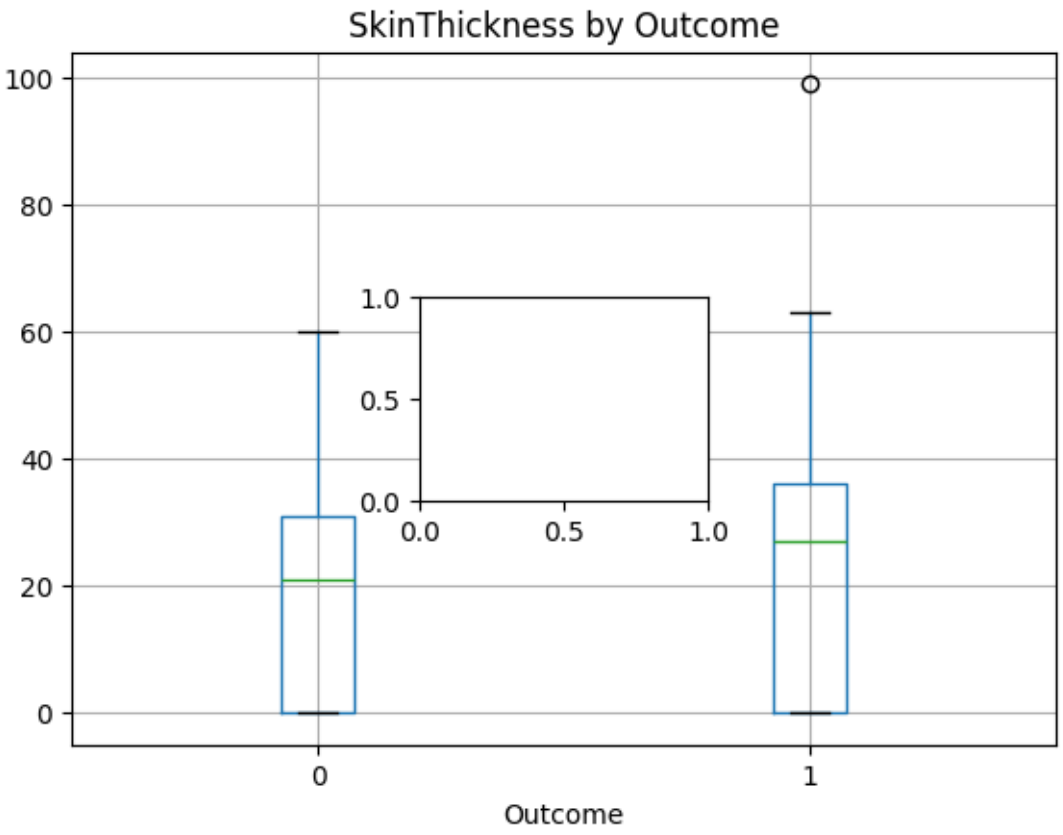
```
        data = diabetes_dataset[diabetes_dataset['Outcome'] == outcome][column]
        plt.hist(data, cumulative=True, density=True, alpha=0.5,
                 label=f'Outcome {outcome}', bins=30)
    plt.title(f'{column} Cumulative Distribution')
    plt.legend()
plt.tight_layout()
plt.show()

# 11. Feature correlations with outcome
correlations = diabetes_dataset.corr()['Outcome'].sort_values(ascending=False)
plt.figure(figsize=(10, 6))
plt.bar(correlations.index, correlations.values)
plt.xticks(rotation=45)
plt.title('Feature Correlations with Diabetes Outcome')
plt.tight_layout()
plt.show()

# 12. Pair-wise scatter plots for most correlated features
top_features = correlations.index[:4]  # Get top 4 correlated features
plt.figure(figsize=(12, 12))
for i, feature1 in enumerate(top_features):
    for j, feature2 in enumerate(top_features):
        plt.subplot(4, 4, i*4 + j + 1)
        if feature1 != feature2:
            plt.scatter(diabetes_dataset[feature1], diabetes_dataset[feature2],
                        c=diabetes_dataset['Outcome'], cmap='coolwarm', alpha=0.
 ↪5)
            plt.xlabel(feature1)
            plt.ylabel(feature2)
        else:
            plt.hist(diabetes_dataset[feature1], bins=30)
            plt.xlabel(feature1)
plt.tight_layout()
plt.show()
```

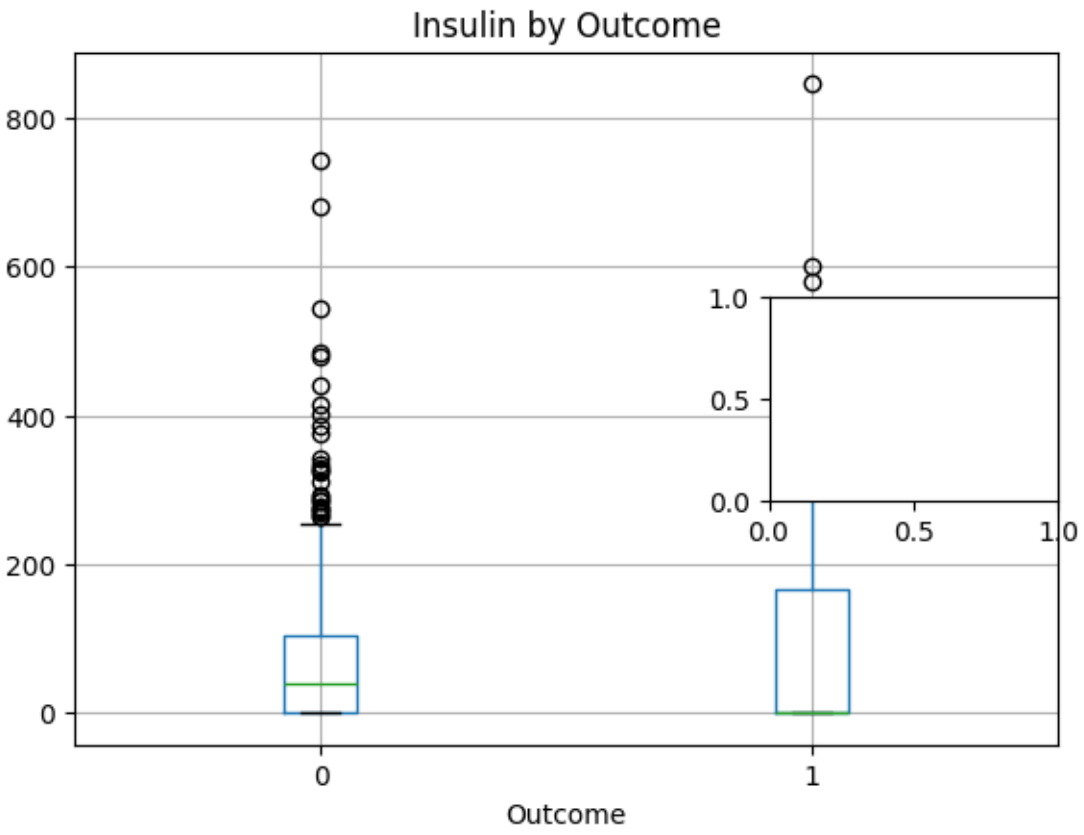

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

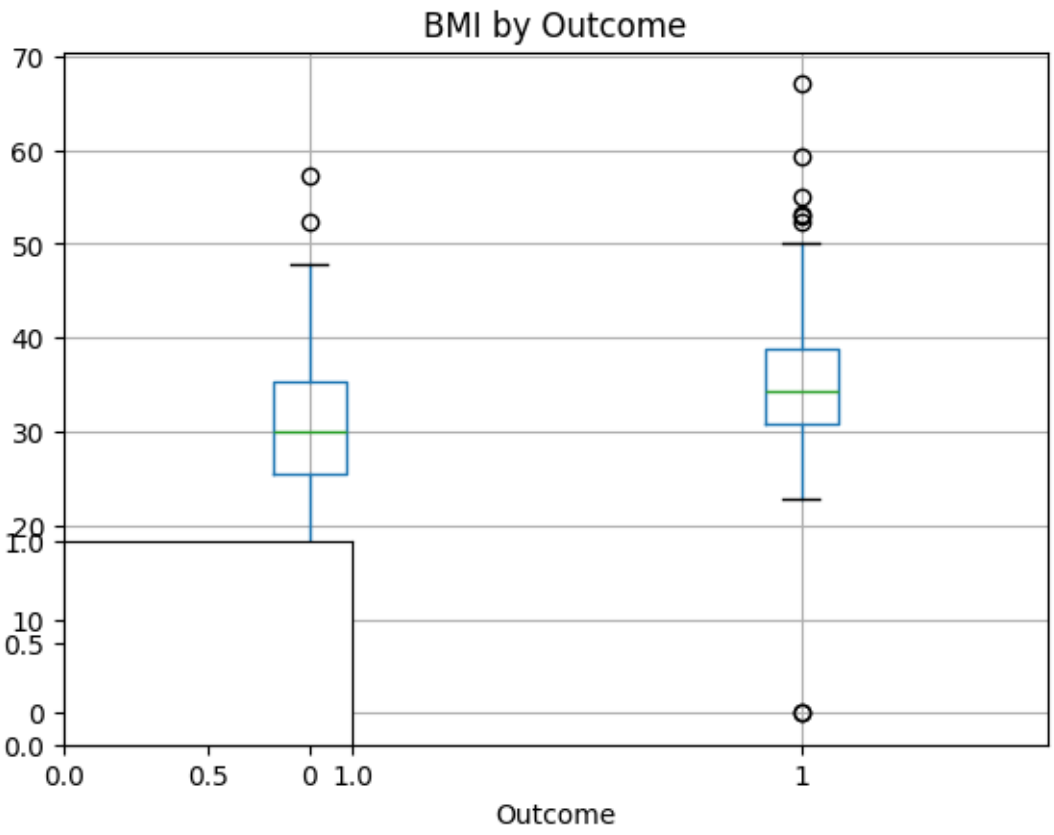

Pregnancies by Outcome
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
1
Outcome
1.0
0.5
0.0
0.0
0.5
1.0

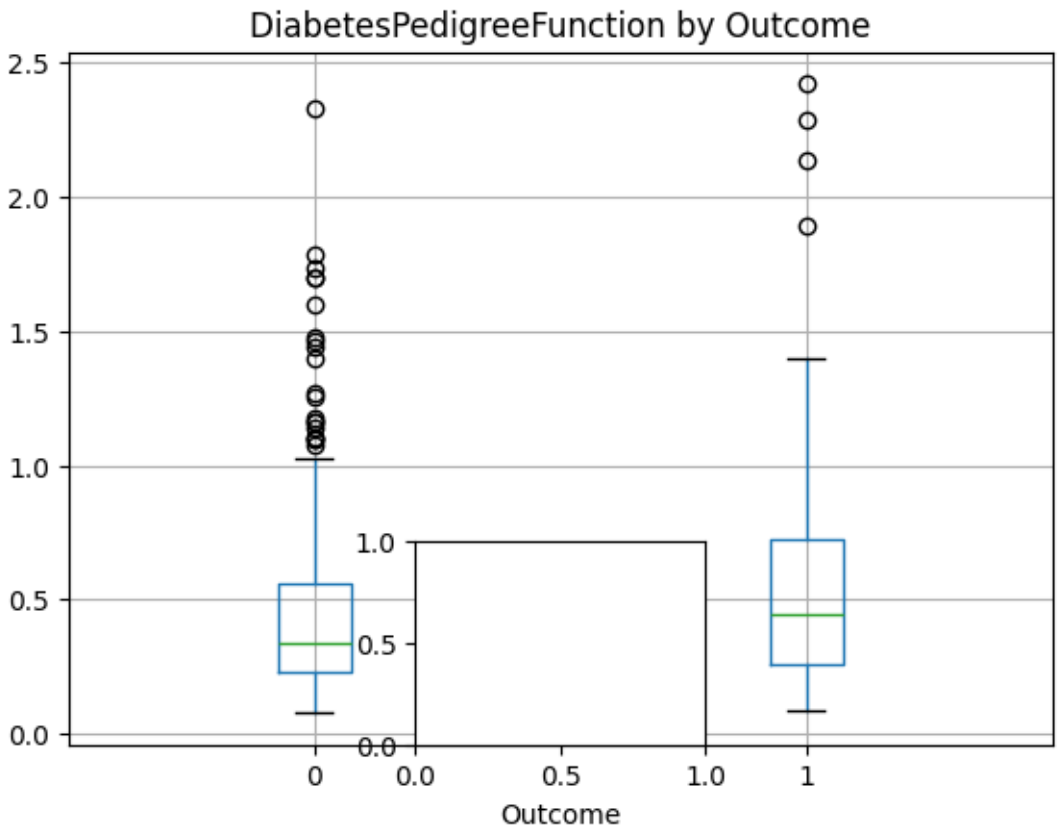

Glucose by Outcome
200
175
150
125
100
75
50
25
0
0
1
Outcome
1.0
0.5
0.0
0.0
0.5
1.0

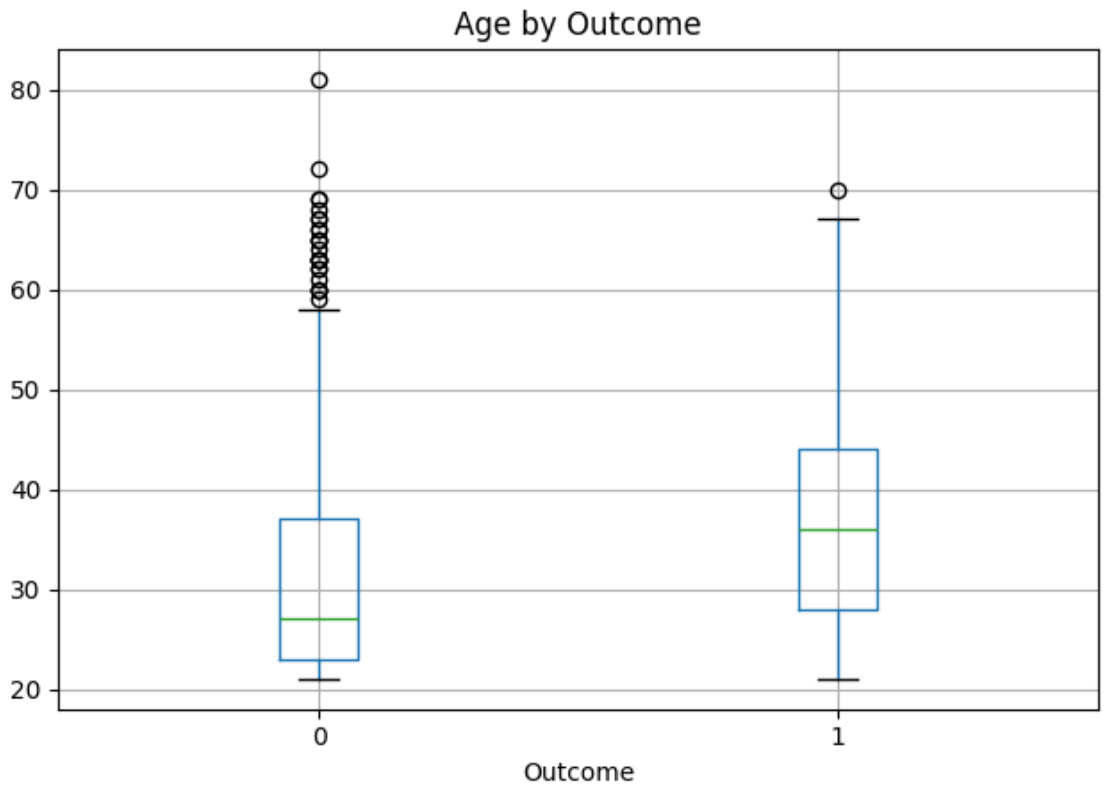

BloodPressure by Outcome
120
100
80
60
40
20
0
0.0
0.5
1.0
0
1
Outcome

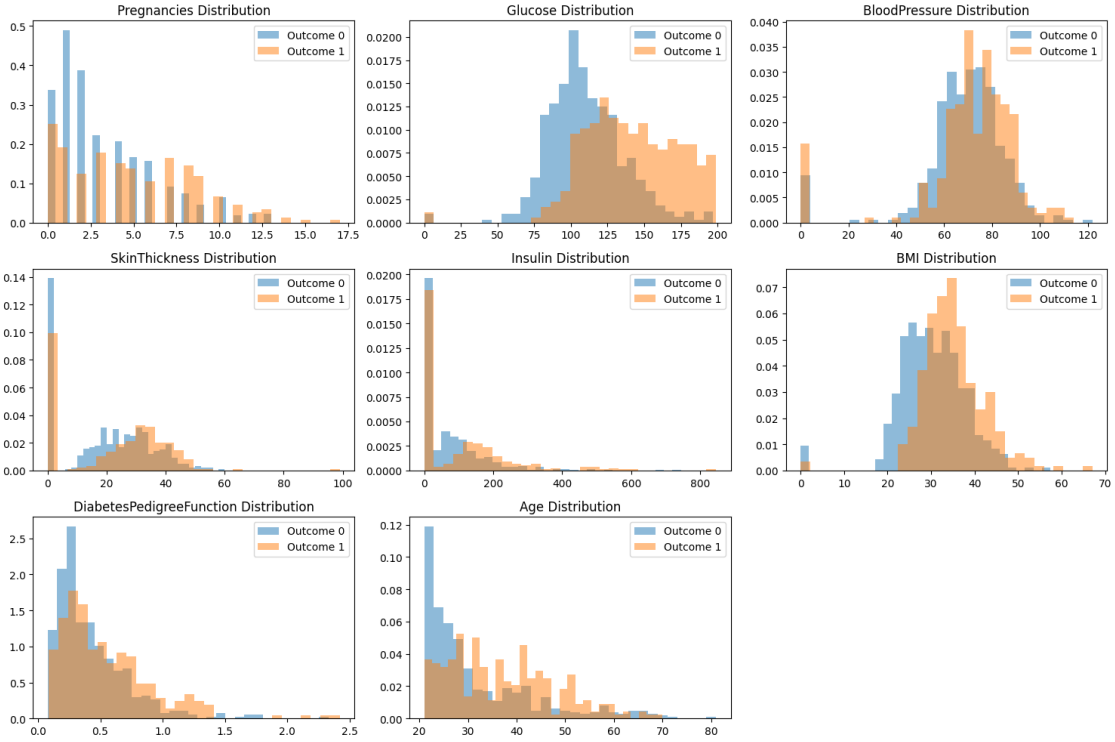

SkinThickness by Outcome
100
80
60
40
20
0
1.0
0.5
0.0
0.0
0.5
1.0
0
1
Outcome


Insulin by Outcome
800
600
400
200
0
1.0
0.5
0.0
0.0
0.5
1.0
0
1
Outcome

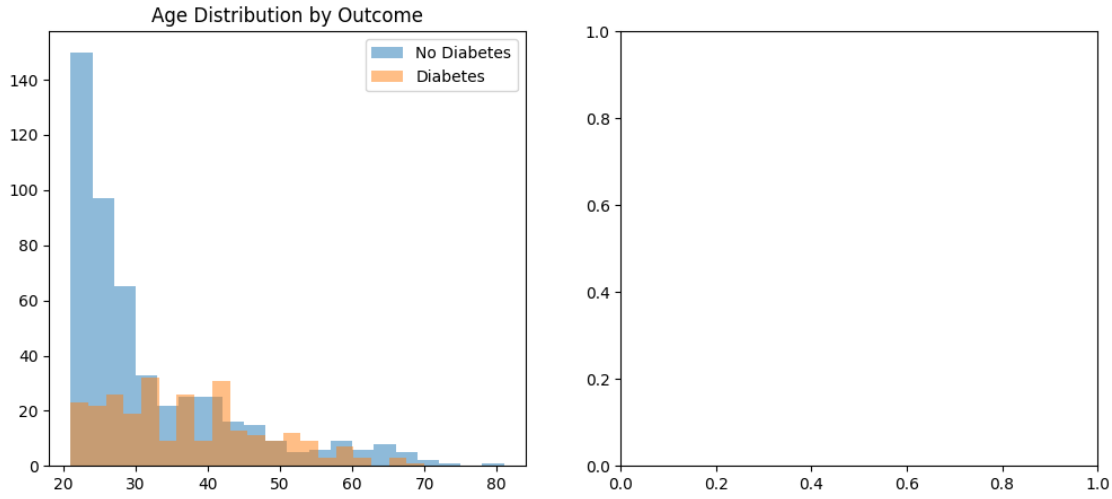

BMI by Outcome
70
60
50
40
30
20
1.0
10
0.5
0
0.0
0.0
0.5
0
1.0
1
Outcome

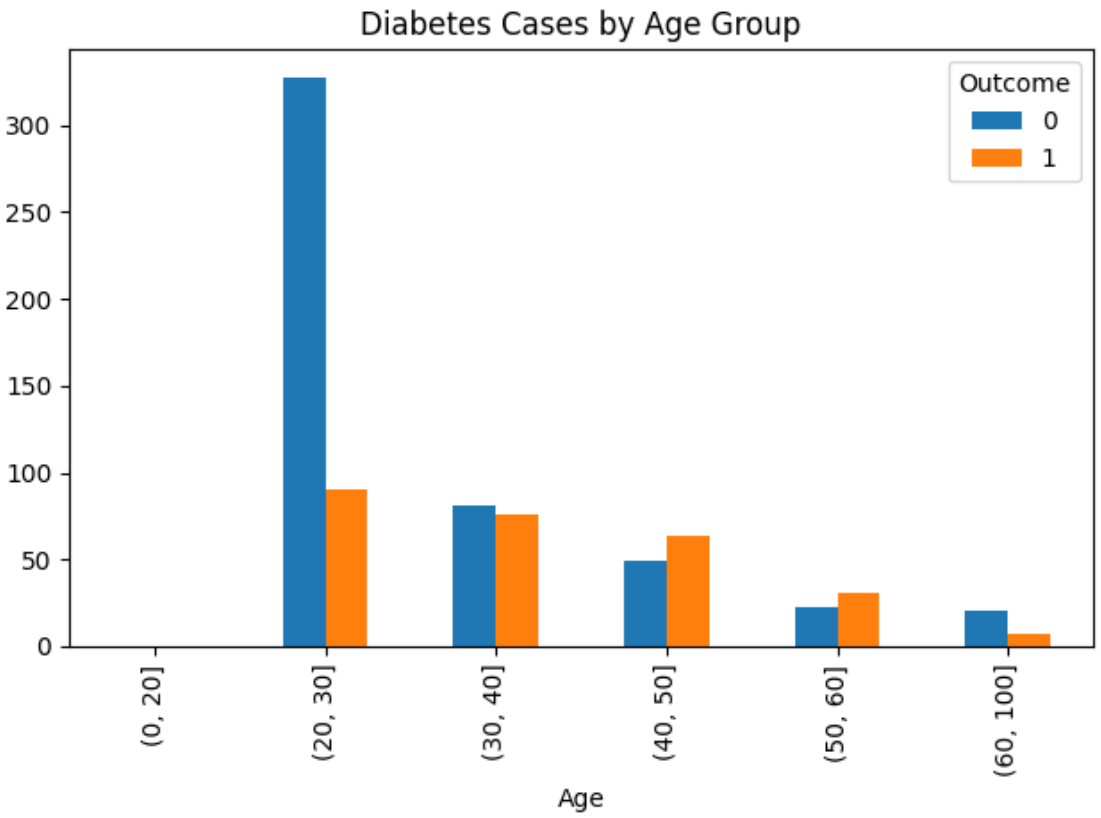

DiabetesPedigreeFunction by Outcome
2.5
2.0
1.5
1.0
0.5
0.0
1.0
0.5
0.0
0
0.0
0.5
1.0
1
Outcome

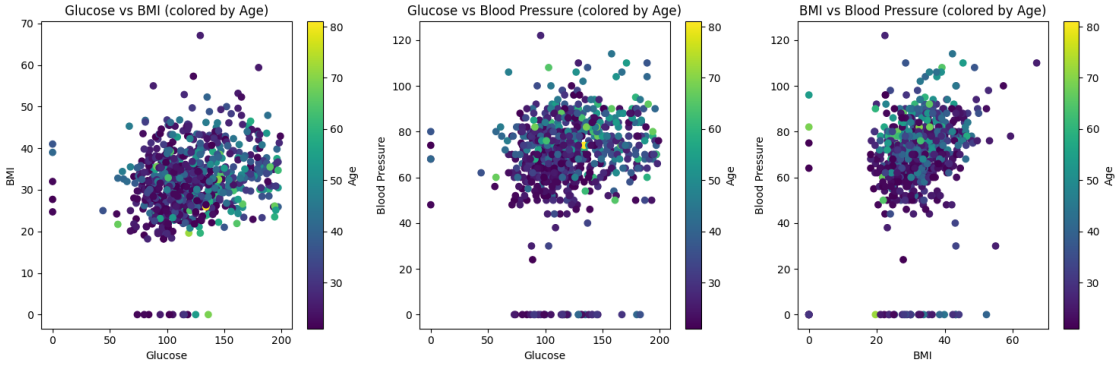

Age by Outcome
80
70
60
50
40
30
20
0
1
Outcome

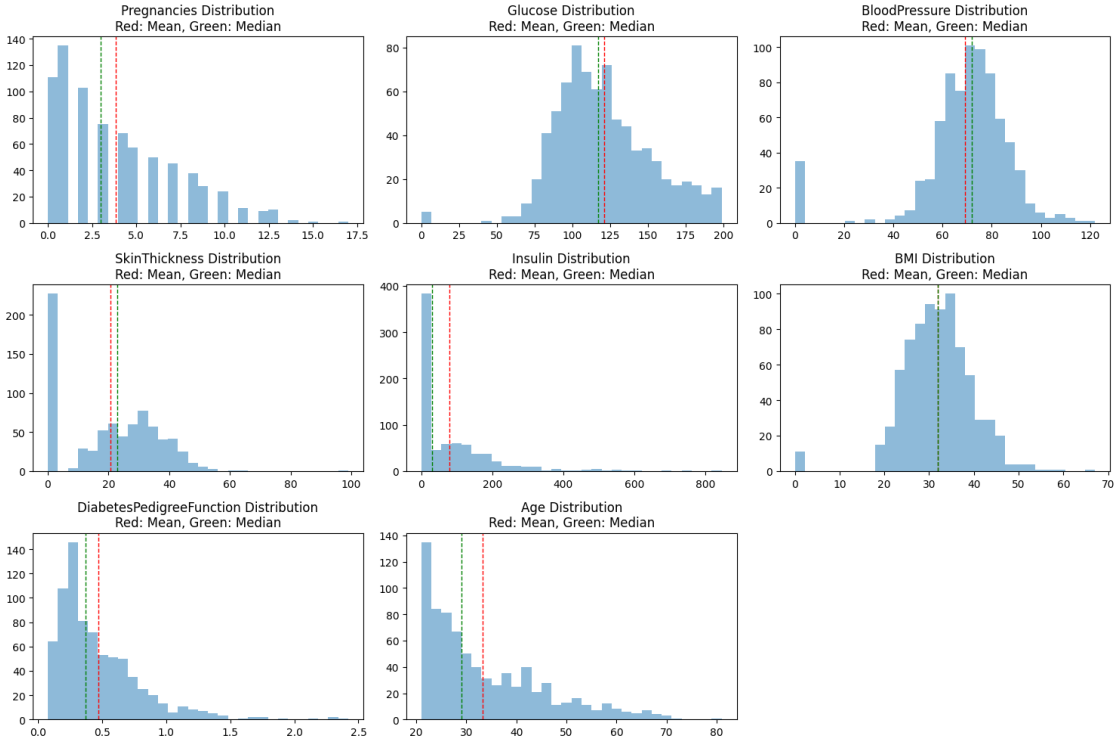

Pregnancies Distribution
Glucose Distribution
BloodPressure Distribution
SkinThickness Distribution
Insulin Distribution
BMI Distribution
DiabetesPedigreeFunction Distribution
Age Distribution
Outcome 0
Outcome 1

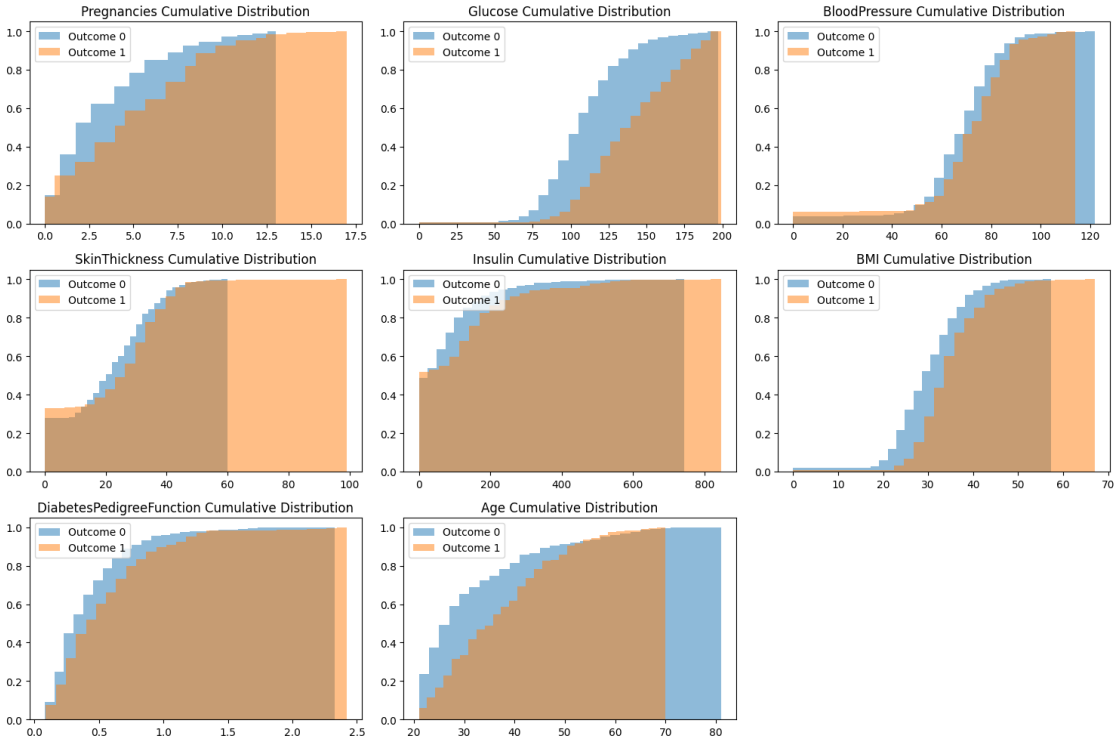
```
C:\Users\KIIT\AppData\Local\Temp\ipykernel_20444\334839278.py:39: FutureWarning:
The default of observed=False is deprecated and will be changed to True in a
future version of pandas. Pass observed=False to retain current behavior or
observed=True to adopt the future default and silence this warning.
  diabetes_dataset.groupby([age_groups,
'Outcome']).size().unstack().plot(kind='bar')
```

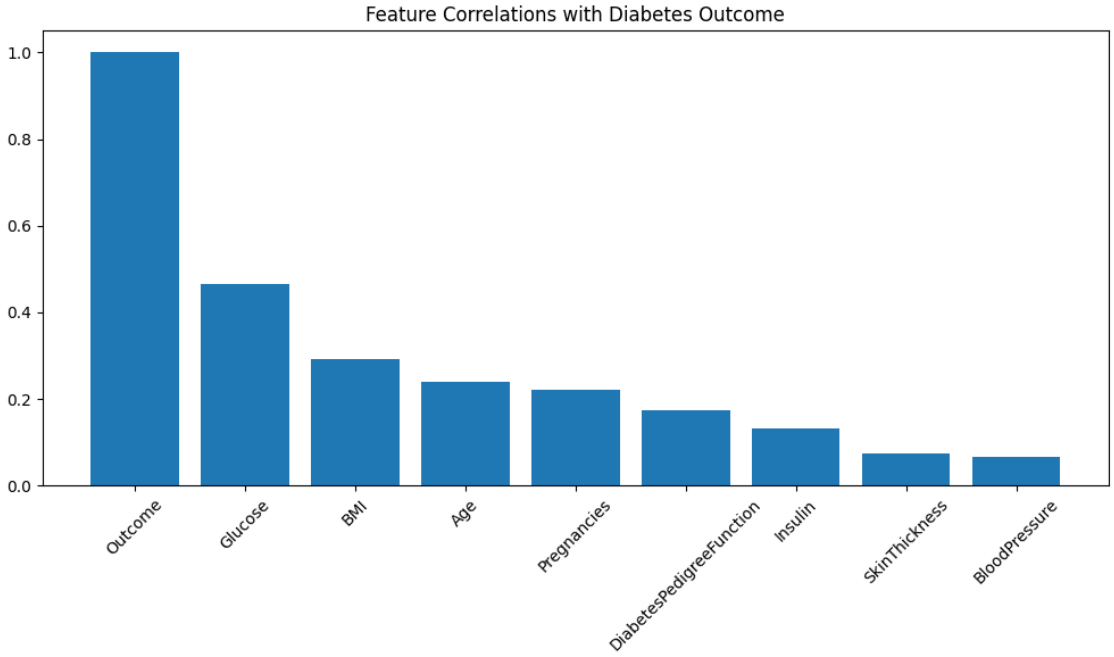


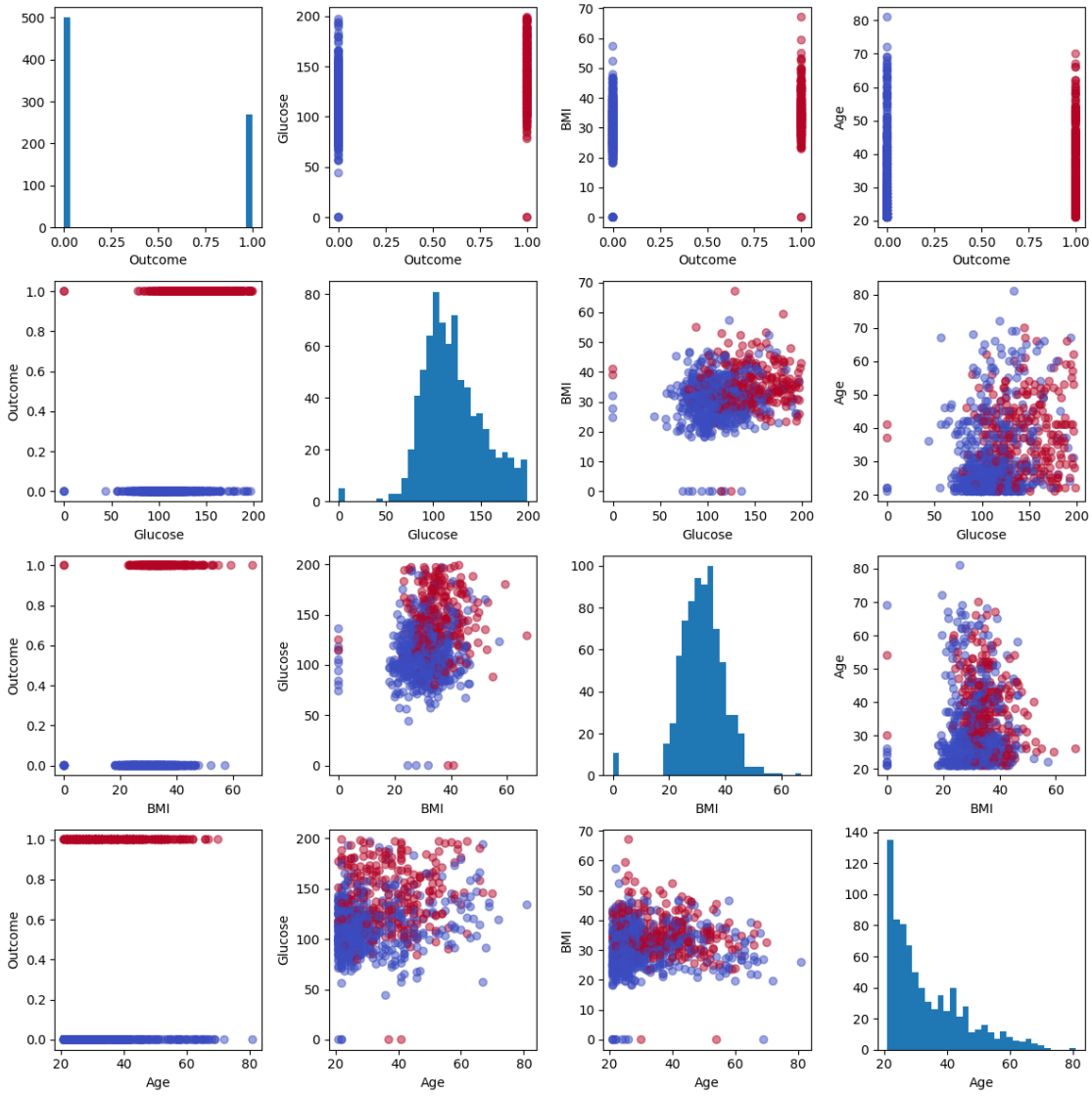

Diabetes Cases by Age Group
Outcome
0
1
300
250
200
150
100
50
0
(0, 20]
(20, 30]
(30, 40]
(40, 50]
(50, 60]
(60, 100]
Age


Glucose vs BMI (colored by Age)
BMI
Glucose
Age
Glucose vs Blood Pressure (colored by Age)
Blood Pressure
Glucose
Age
BMI vs Blood Pressure (colored by Age)
Blood Pressure
BMI
Age


Pregnancies Distribution
Red: Mean, Green: Median
Glucose Distribution
Red: Mean, Green: Median
BloodPressure Distribution
Red: Mean, Green: Median
SkinThickness Distribution
Red: Mean, Green: Median
Insulin Distribution
Red: Mean, Green: Median
BMI Distribution
Red: Mean, Green: Median
DiabetesPedigreeFunction Distribution
Red: Mean, Green: Median
Age Distribution
Red: Mean, Green: Median


Pregnancies Cumulative Distribution
Glucose Cumulative Distribution
BloodPressure Cumulative Distribution
SkinThickness Cumulative Distribution
Insulin Cumulative Distribution
BMI Cumulative Distribution
DiabetesPedigreeFunction Cumulative Distribution
Age Cumulative Distribution
Outcome 0
Outcome 1


Feature Correlations with Diabetes Outcome
1.0
0.8
0.6
0.4
0.2
0.0
Outcome
Glucose
BMI
Age
Pregnancies
DiabetesPedigreeFunction
Insulin
SkinThickness
BloodPressure


Outcome
Glucose
BMI
Age