wjcb86tlq

February 18, 2024

[ ]:
```
pip install --upgrade scikit-learn
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-
packages (1.2.2)
Collecting scikit-learn
  Downloading
scikit_learn-1.4.0-1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(12.1 MB)
                           12.1/12.1 MB
35.3 MB/s eta 0:00:00
Requirement already satisfied: numpy<2.0,>=1.19.5 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.23.5)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.2.0)
Installing collected packages: scikit-learn
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.2.2
    Uninstalling scikit-learn-1.2.2:
      Successfully uninstalled scikit-learn-1.2.2
Successfully installed scikit-learn-1.4.0
```

[ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

[ ]:
```
data = pd.read_csv("/content/diabetes.csv")
display(data)
```

```
     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  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
..                        ...  ...      ...
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]
```

[ ]:
```
data.describe()
```

[ ]:
```
       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
```

[ ]:
```
data.isnull().sum()
```

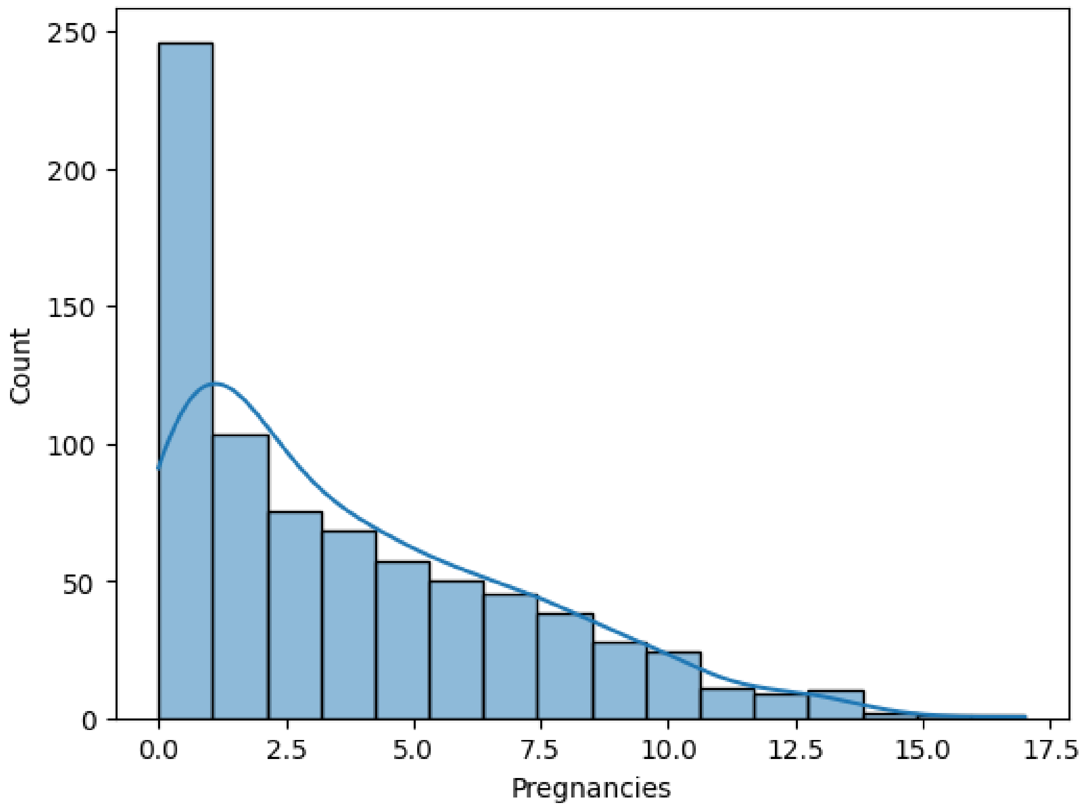
[ ]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

[ ]:
```python
data.median()
```

[ ]:
```
Pregnancies                   3.0000
Glucose                     117.0000
BloodPressure                72.0000
SkinThickness                23.0000
Insulin                      30.5000
BMI                          32.0000
DiabetesPedigreeFunction      0.3725
Age                          29.0000
Outcome                       0.0000
dtype: float64
```

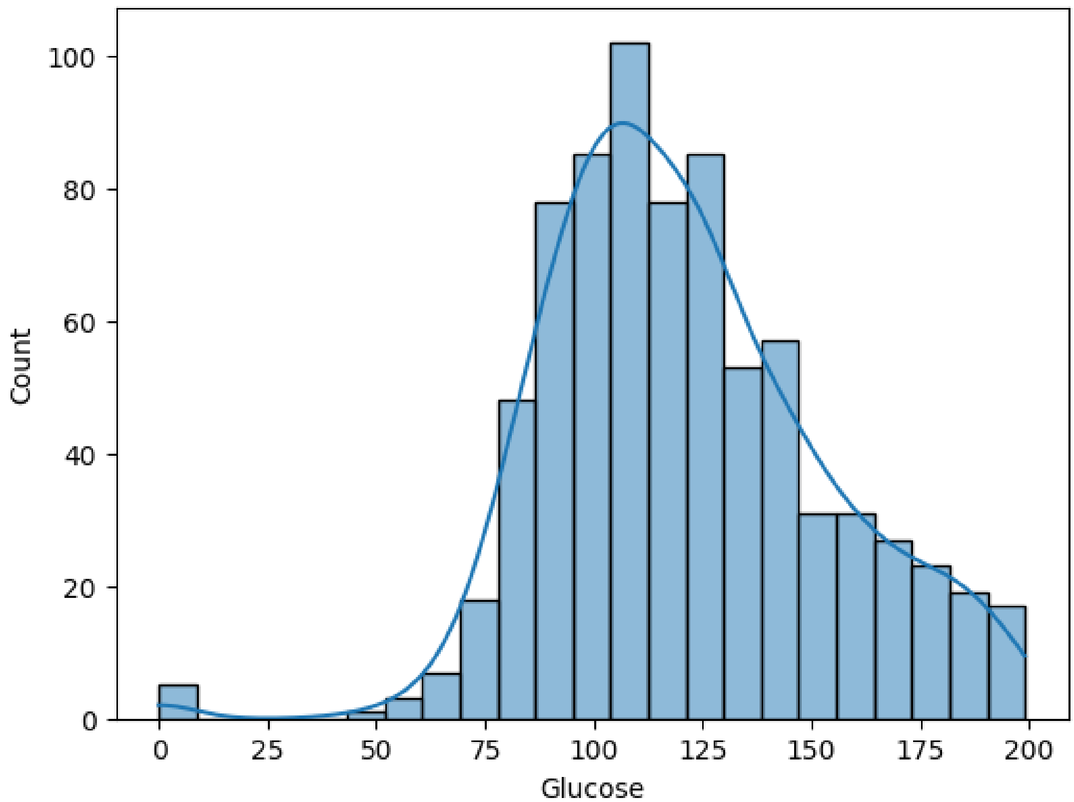
[ ]:
```python
sns.histplot(data = data, x = 'Pregnancies', kde = True)
```

[ ]:
```
<Axes: xlabel='Pregnancies', ylabel='Count'>
```



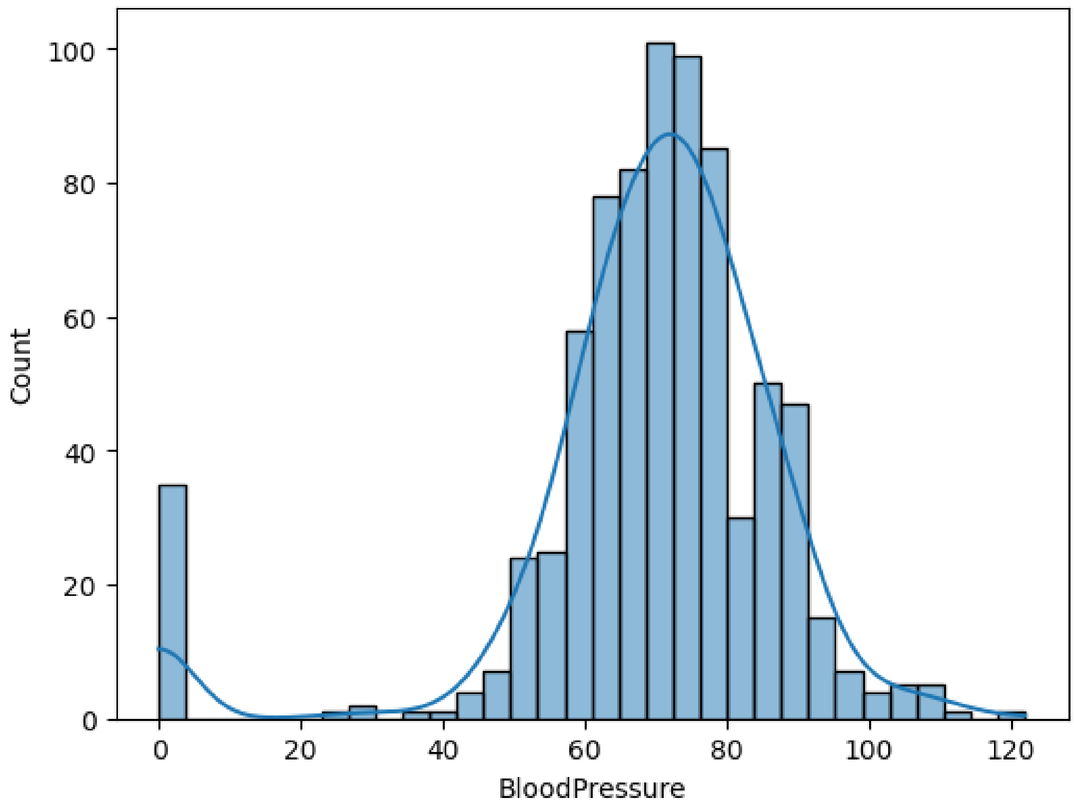
[ ]:
```
sns.histplot(data = data, x = 'Glucose', kde = True)
```

[ ]: <Axes: xlabel='Glucose', ylabel='Count'>



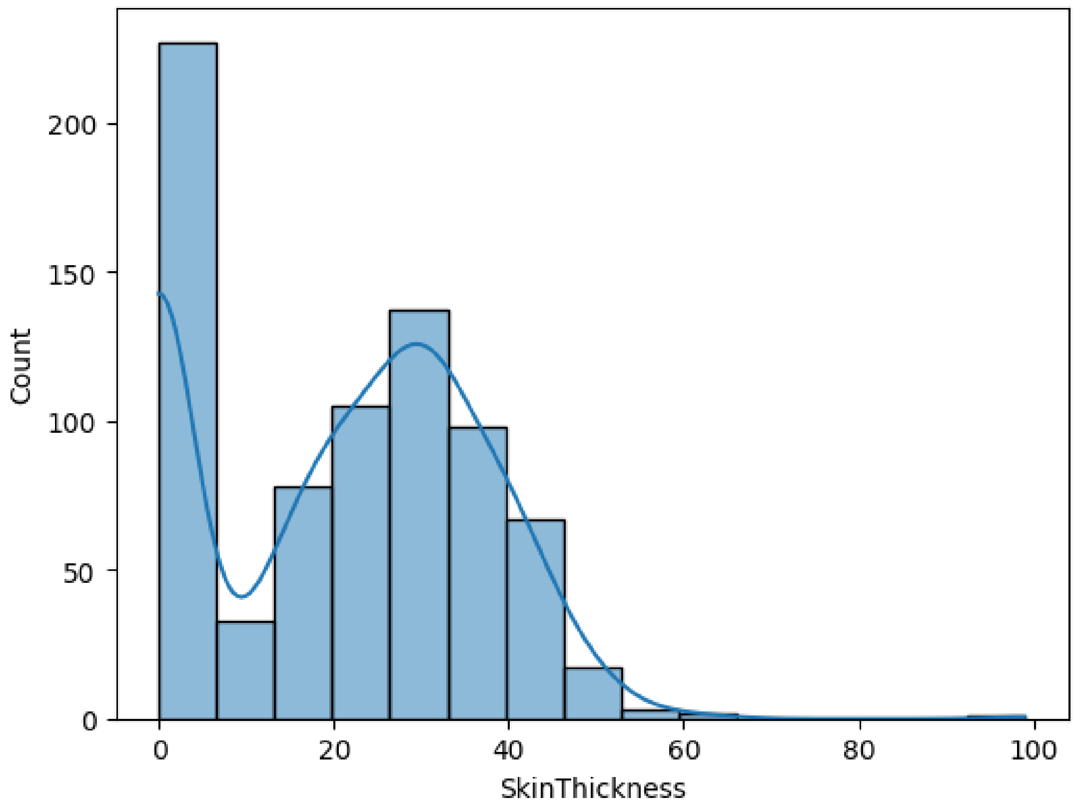
[ ]:
```
sns.histplot(data = data, x = 'BloodPressure', kde = True)
```

[ ]: <Axes: xlabel='BloodPressure', ylabel='Count'>



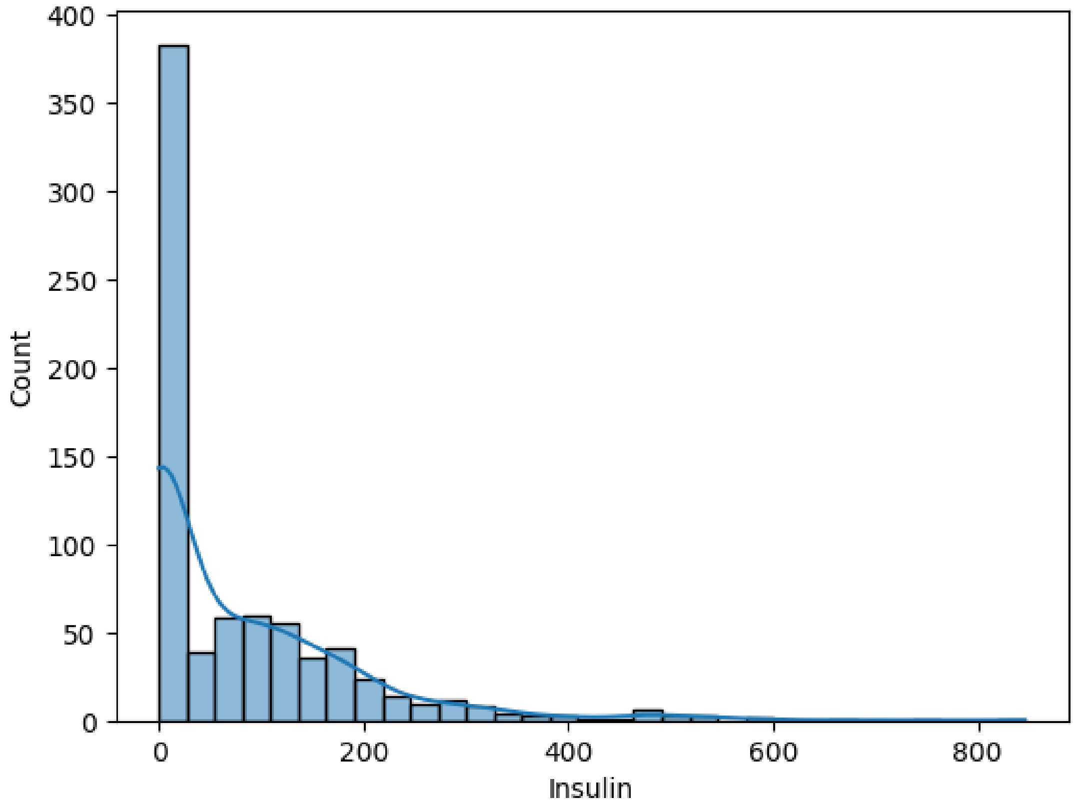
[ ]:
```
sns.histplot(data = data, x = 'SkinThickness', kde = True)
```

[ ]: <Axes: xlabel='SkinThickness', ylabel='Count'>



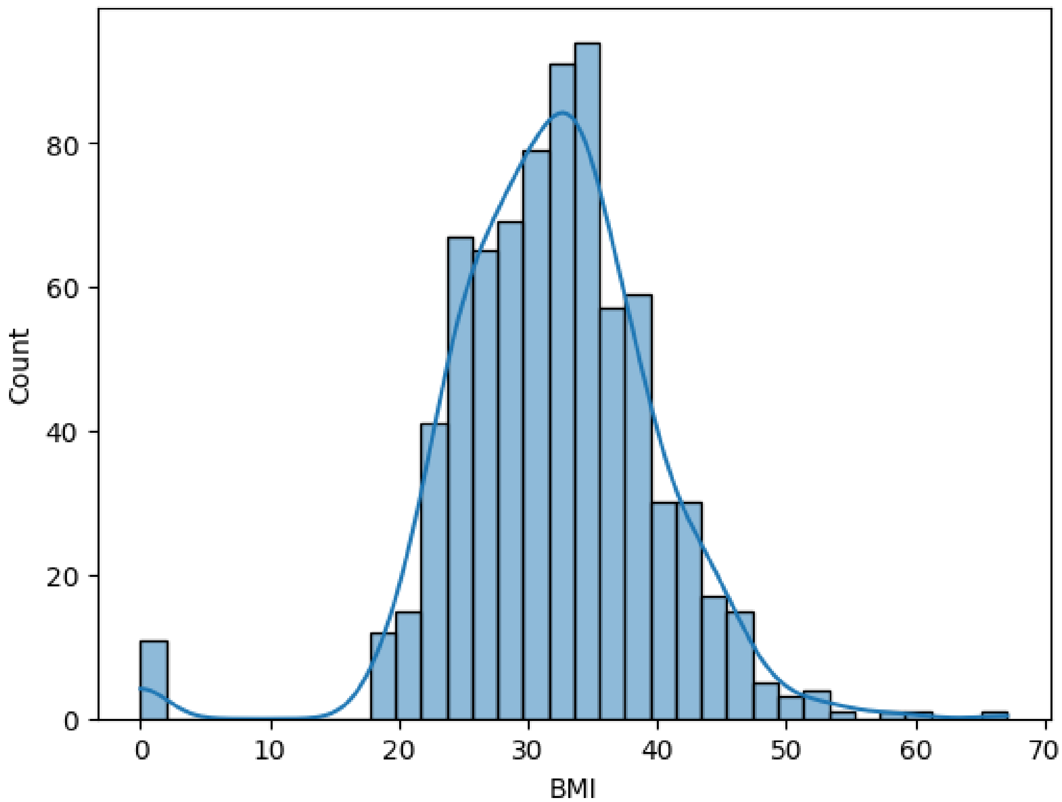
[ ]:
```
sns.histplot(data = data, x = 'Insulin', kde = True)
```

[ ]: <Axes: xlabel='Insulin', ylabel='Count'>



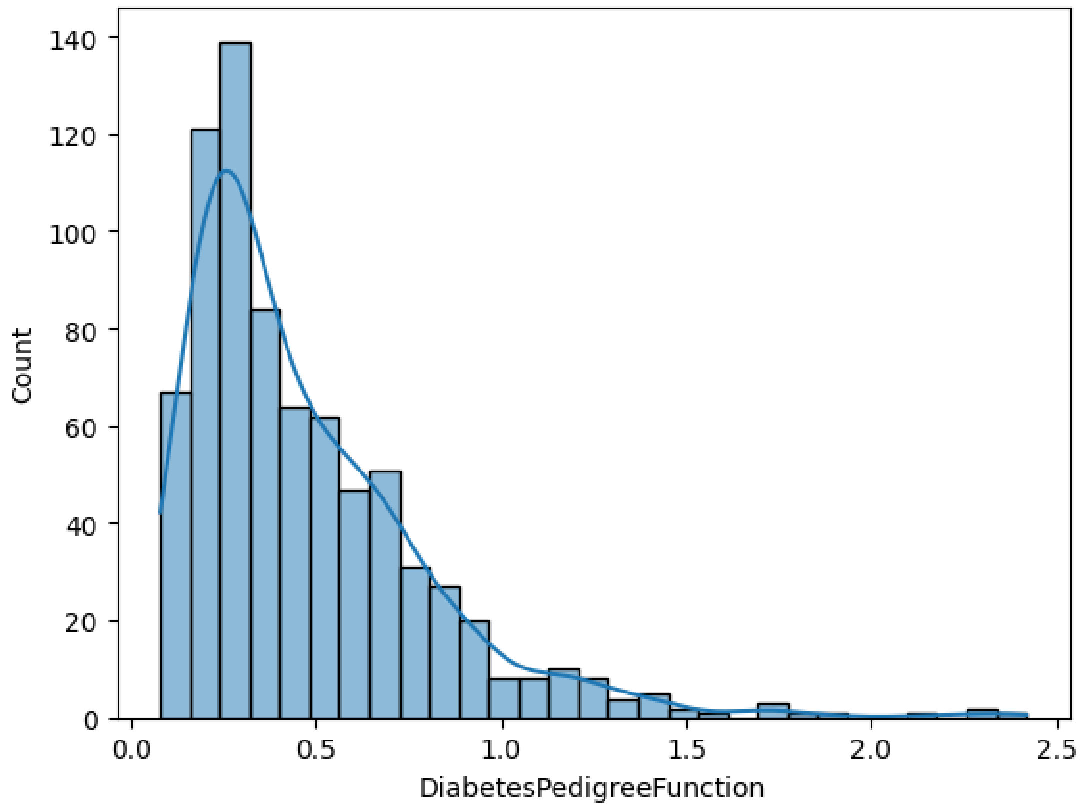
[ ]:
```
sns.histplot(data = data, x = 'BMI', kde = True)
```

[ ]: <Axes: xlabel='BMI', ylabel='Count'>



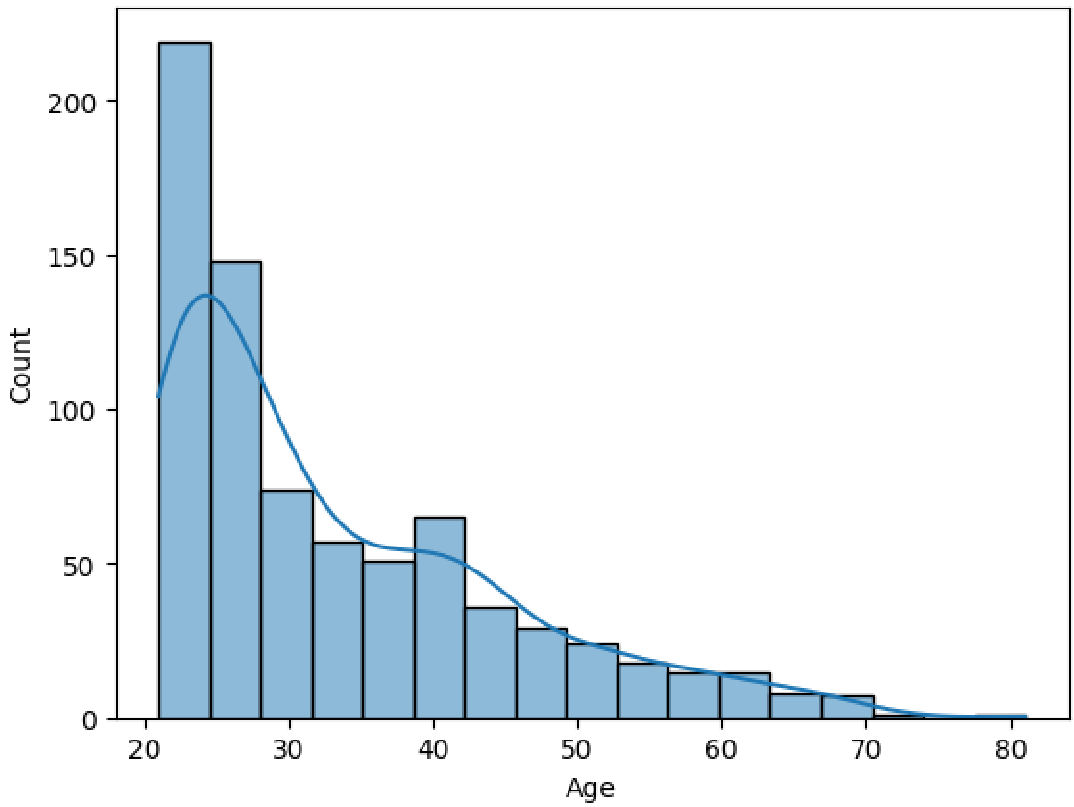
[ ]:
```
sns.histplot(data = data, x = 'DiabetesPedigreeFunction', kde = True)
```

[ ]: <Axes: xlabel='DiabetesPedigreeFunction', ylabel='Count'>



[ ]:
```
sns.histplot(data = data, x = 'Age', kde = True)
```

[ ]: <Axes: xlabel='Age', ylabel='Count'>




```
[ ]: data[data['Outcome'] == 0].count()
```

```
[ ]: Pregnancies                 500
     Glucose                     500
     BloodPressure               500
     SkinThickness               500
     Insulin                     500
     BMI                         500
     DiabetesPedigreeFunction    500
     Age                         500
     Outcome                     500
     dtype: int64
```

```
[ ]: data[data['Outcome'] == 1].count()
```

```
[ ]: Pregnancies                 268
     Glucose                     268
     BloodPressure               268
     SkinThickness               268
     Insulin                     268
     BMI                         268
```

```
DiabetesPedigreeFunction    268
Age                         268
Outcome                     268
dtype: int64
```

[ ]:
```
from sklearn.utils import resample

majority_class = data[data['Outcome'] == 0]
minority_class = data[data['Outcome'] == 1]
minority_upsampled = resample(minority_class, replace=True,␣
 ↪n_samples=len(majority_class), random_state=42)
data_balanced = pd.concat([majority_class, minority_upsampled], ignore_index =␣
 ↪True)
display(data_balanced)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              1       85             66             29        0  26.6
1              1       89             66             23       94  28.1
2              5      116             74              0        0  25.6
3             10      115              0              0        0  35.3
4              4      110             92              0        0  37.6
..           …        …            …             …       …    …
995            7      168             88             42      321  38.2
996            8      143             66              0        0  34.9
997            3      130             78             23       79  28.4
998            6      115             60             39        0  33.7
999            4      184             78             39      277  37.0

     DiabetesPedigreeFunction  Age  Outcome
0                       0.351   31        0
1                       0.167   21        0
2                       0.201   30        0
3                       0.134   29        0
4                       0.191   30        0
..                        …   …      …
995                     0.787   40        1
996                     0.129   41        1
997                     0.323   34        1
998                     0.245   40        1
999                     0.264   31        1

[1000 rows x 9 columns]
```

[ ]:
```
data_balanced[data_balanced['Outcome'] == 0].count()
```

[ ]:
```
Pregnancies                 500
Glucose                     500
```

```
BloodPressure               500
SkinThickness               500
Insulin                     500
BMI                         500
DiabetesPedigreeFunction    500
Age                         500
Outcome                     500
dtype: int64
```

[ ]:
```
data_balanced[data_balanced['Outcome'] == 1].count()
```

[ ]:
```
Pregnancies                 500
Glucose                     500
BloodPressure               500
SkinThickness               500
Insulin                     500
BMI                         500
DiabetesPedigreeFunction    500
Age                         500
Outcome                     500
dtype: int64
```

[ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.model_selection import train_test_split, KFold, cross_val_score,␣
 ↪RandomizedSearchCV, ValidationCurveDisplay
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, classification_report, confusion_matrix
```

[ ]:
```
y = data['Outcome']
y_b = data_balanced['Outcome']
data = data.drop('Outcome', axis = 1)
data_balanced = data_balanced.drop('Outcome', axis = 1)
```

[ ]:
```
std = StandardScaler()
data = std.fit_transform(data)
data_balanced = std.fit_transform(data_balanced)
```

[ ]:
```
display(data)
```

```
array([[ 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]])
```

[ ]:
```
display(data_balanced)
```

```
array([[-0.89511603, -1.25513659, -0.21517238, …, -0.77277546,
        -0.38393058, -0.28171954],
       [-0.89511603, -1.12702548, -0.21517238, …, -0.57484782,
        -0.94627816, -1.14245752],
       [ 0.22693477, -0.26227547,  0.20102178, …, -0.90472721,
        -0.84236611, -0.36779334],
       …,
       [-0.33409063,  0.18611342,  0.40911886, …, -0.5352623 ,
        -0.46950522, -0.02349815],
       [ 0.50744747, -0.29430325, -0.52731801, …,  0.16408201,
        -0.70789169,  0.49294464],
       [-0.05357793,  1.91561342,  0.40911886, …,  0.5995228 ,
        -0.64982319, -0.28171954]])
```

[ ]:
```
n = int(input())
poly = PolynomialFeatures(degree = n)
data_poly = poly.fit_transform(data)
data_balanced_poly = poly.fit_transform(data_balanced)
```

```
2
```

[ ]:
```
print(data_poly.shape)
print(data_balanced_poly.shape)
```

```
(768, 45)
(1000, 45)
```

[ ]:
```
display(data_poly)
```

```
array([[ 1.        ,  0.63994726,  0.84832379, …,  0.21948473,
         0.66806741,  2.03346289],
       [ 1.        , -0.84488505, -1.12339636, …,  0.13326937,
         0.06960683,  0.03635578],
       [ 1.        ,  1.23388019,  1.94372388, …,  0.36529612,
        -0.06381478,  0.01114801],
       …,
       [ 1.        ,  0.3429808 ,  0.00330087, …,  0.46948994,
         0.18894869,  0.07604339],
       [ 1.        , -0.84488505,  0.1597866 , …,  0.13771596,
```

```
        -0.43445989,  1.37061376],
       [ 1.        , -0.84488505, -0.8730192 , …,  0.22447227,
         0.41284394,  0.75929253]])
```

[ ]:
```
display(data_balanced_poly)
```

```
array([[ 1.00000000e+00, -8.95116026e-01, -1.25513659e+00, …,
         1.47402694e-01,  1.08160748e-01,  7.93658993e-02],
       [ 1.00000000e+00, -8.95116026e-01, -1.12702548e+00, …,
         8.95442364e-01,  1.08108260e+00,  1.30520918e+00],
       [ 1.00000000e+00,  2.26934774e-01, -2.62275473e-01, …,
         7.09580666e-01,  3.09816644e-01,  1.35271939e-01],
       …,
       [ 1.00000000e+00, -3.34090626e-01,  1.86113417e-01, …,
         2.20435148e-01,  1.10325025e-02,  5.52162903e-04],
       [ 1.00000000e+00,  5.07447475e-01, -2.94303251e-01, …,
         5.01110645e-01, -3.48951414e-01,  2.42994418e-01],
       [ 1.00000000e+00, -5.35779257e-02,  1.91561342e+00, …,
         4.22270179e-01,  1.83067890e-01,  7.93658993e-02]])
```

[ ]:
```
data = np.tile(data, (10,1))
data_balanced = np.tile(data_balanced, (10,1))
data_poly = np.tile(data_poly, (10,1))
data_balanced_poly = np.tile(data_balanced_poly, (10,1))
y = np.tile(y, 10)
y_b = np.tile(y_b, 10)
```

[ ]:
```
x_train, x_temp, y_train, y_temp = train_test_split(data, y, test_size=0.4,␣
 ↪random_state=42, stratify = y)
x_val, x_test, y_val, y_test = train_test_split(x_temp, y_temp, test_size=0.5,␣
 ↪random_state=42, stratify = y_temp)
x_train_b, x_temp_b, y_train_b, y_temp_b = train_test_split(data_balanced, y_b,␣
 ↪test_size=0.4, random_state=42, stratify = y_b)
x_val_b, x_test_b, y_val_b, y_test_b = train_test_split(x_temp_b, y_temp_b,␣
 ↪test_size=0.5, random_state=42, stratify = y_temp_b) #b for balanced
x_train_p, x_temp_p, y_train_p, y_temp_p = train_test_split(data_poly, y,␣
 ↪test_size=0.4, random_state=42, stratify = y)
x_val_p, x_test_p, y_val_p, y_test_p = train_test_split(x_temp_p, y_temp_p,␣
 ↪test_size=0.5, random_state=42, stratify = y_temp_p)
x_train_b_p, x_temp_b_p, y_train_b_p, y_temp_b_p =␣
 ↪train_test_split(data_balanced_poly, y_b, test_size=0.4, random_state=42,␣
 ↪stratify = y_b)
x_val_b_p, x_test_b_p, y_val_b_p, y_test_b_p = train_test_split(x_temp_b_p,␣
 ↪y_temp_b_p, test_size=0.5, random_state=42, stratify = y_temp_b_p) #p for␣
 ↪polygonal features
```

[ ]:
```
print(x_train.shape, x_test.shape, x_val.shape)
print(x_train_b.shape, x_test_b.shape, x_val_b.shape)
print(x_train_p.shape, x_test_p.shape, x_val_p.shape)
print(x_train_b_p.shape, x_test_b_p.shape, x_val_b_p.shape)
```

```
(460, 8) (154, 8) (154, 8)
(600, 8) (200, 8) (200, 8)
(460, 45) (154, 45) (154, 45)
(600, 45) (200, 45) (200, 45)
```

[ ]:
```
model_logreg = LogisticRegression(random_state = 42, max_iter = 10000)
```

[ ]:
```
def kfold(input_data, output_data):
    kf = KFold( n_splits = int(input_data.shape[0]/10), shuffle = True,␣
 ↪random_state = 42)
    scores = cross_val_score(model_logreg, input_data, output_data, cv = kf)
    print("Cross validation scores : ", scores)
    print("Mean cross validation score : ", np.mean(scores))
```

[ ]:
```
kfold(x_train, y_train)
kfold(x_train_b, y_train_b)
kfold(x_train_p, y_train_p)
kfold(x_train_b_p, y_train_b_p)
```

```
Cross validation scores :  [1.  0.7 0.7 0.8 0.9 0.7 0.6 0.7 0.9 0.8 0.8 0.8 0.9
0.9 0.7 0.9 0.9 0.7
 0.9 0.8 0.8 0.8 0.7 0.5 0.8 0.5 0.6 0.7 0.8 1.  0.9 0.6 0.7 0.8 1.  1.
 0.8 1.  0.8 0.9 0.7 0.8 0.8 0.6 0.8 0.7]
Mean cross validation score :  0.7869565217391304
Cross validation scores :  [0.8 0.4 0.8 0.7 0.6 0.6 0.7 0.8 0.8 0.7 0.6 0.7 0.9
1.  0.7 0.7 0.8 0.8
 0.8 0.6 0.7 0.8 0.7 0.9 0.6 0.6 0.7 0.8 0.5 0.6 0.6 0.6 0.7 0.8 0.6 0.8
 0.9 0.8 0.7 0.9 0.7 0.8 0.6 0.9 0.7 0.8 0.7 0.9 0.9 0.7 0.7 0.9 0.8 0.4
 0.6 0.9 0.6 0.9 0.8 0.9]
Mean cross validation score :  0.7333333333333333
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-425-391bbad65a22> in <cell line: 3>()
      1 kfold(x_train, y_train)
      2 kfold(x_train_b, y_train_b)
----> 3 kfold(x_train_p, y_train_p)
      4 kfold(x_train_b_p, y_train_b_p)

<ipython-input-424-50f5deeac830> in kfold(input_data, output_data)
      1 def kfold(input_data, output_data):
```

```
      2     kf = KFold( n_splits = int(input_data.shape[0]/10), shuffle = True, random_state = 42)
----> 3     scores = cross_val_score(model_logreg, input_data, output_data, cv = kf)
      4     print("Cross validation scores : ", scores)
      5     print("Mean cross validation score : ", np.mean(scores))

/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validation.py in wrapper(*args, **kwargs)
    211                     )
    212                 ):
--> 213                     return func(*args, **kwargs)
    214             except InvalidParameterError as e:
    215                 # When the function is just a wrapper around an estimator, we allow

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py in cross_val_score(estimator, X, y, groups, scoring, cv, n_jobs, verbose, fit_params, params, pre_dispatch, error_score)
    712     scorer = check_scoring(estimator, scoring=scoring)
    713
--> 714     cv_results = cross_validate(
    715         estimator=estimator,
    716         X=X,

/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validation.py in wrapper(*args, **kwargs)
    211                     )
    212                 ):
--> 213                     return func(*args, **kwargs)
    214             except InvalidParameterError as e:
    215                 # When the function is just a wrapper around an estimator, we allow

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py in cross_validate(estimator, X, y, groups, scoring, cv, n_jobs, verbose, fit_params, params, pre_dispatch, return_train_score, return_estimator, return_indices, error_score)
    423     # independent, and that it is pickle-able.
    424     parallel = Parallel(n_jobs=n_jobs, verbose=verbose, pre_dispatch=pre_dispatch)
--> 425     results = parallel(
    426         delayed(_fit_and_score)(
    427             clone(estimator),

/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py in __call__(self, iterable)
```

```
     65             for delayed_func, args, kwargs in iterable
     66         )
---> 67         return super().__call__(iterable_with_config)
     68
     69

/usr/local/lib/python3.10/dist-packages/joblib/parallel.py in __call__(self,␣
↪iterable)
   1861             output = self._get_sequential_output(iterable)
   1862             next(output)
-> 1863             return output if self.return_generator else list(output)
   1864
   1865         # Let's create an ID that uniquely identifies the current call.␣
↪If the

/usr/local/lib/python3.10/dist-packages/joblib/parallel.py in␣
↪_get_sequential_output(self, iterable)
   1790                 self.n_dispatched_batches += 1
   1791                 self.n_dispatched_tasks += 1
-> 1792                 res = func(*args, **kwargs)
   1793                 self.n_completed_tasks += 1
   1794                 self.print_progress()

/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py in␣
↪__call__(self, *args, **kwargs)
    127             config = {}
    128         with config_context(**config):
--> 129             return self.function(*args, **kwargs)

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py␣
↪in _fit_and_score(estimator, X, y, scorer, train, test, verbose, parameters,␣
↪fit_params, score_params, return_train_score, return_parameters,␣
↪return_n_test_samples, return_times, return_estimator, split_progress,␣
↪candidate_progress, error_score)
    888             estimator.fit(X_train, **fit_params)
    889         else:
--> 890             estimator.fit(X_train, y_train, **fit_params)
    891
    892     except Exception:

/usr/local/lib/python3.10/dist-packages/sklearn/base.py in wrapper(estimator,␣
↪*args, **kwargs)
   1349                 )
   1350             ):
-> 1351                 return fit_method(estimator, *args, **kwargs)
   1352
   1353         return wrapper
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py in␣
↪fit(self, X, y, sample_weight)
   1294                 n_threads = 1
   1295
-> 1296         fold_coefs_ = Parallel(n_jobs=self.n_jobs, verbose=self.verbose,␣
↪prefer=prefer)(
   1297             path_func(
   1298                 X,

/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py in␣
↪__call__(self, iterable)
     65             for delayed_func, args, kwargs in iterable
     66         )
---> 67         return super().__call__(iterable_with_config)
     68
     69

/usr/local/lib/python3.10/dist-packages/joblib/parallel.py in __call__(self,␣
↪iterable)
   1861             output = self._get_sequential_output(iterable)
   1862             next(output)
-> 1863             return output if self.return_generator else list(output)
   1864
   1865         # Let's create an ID that uniquely identifies the current call.␣
↪If the

/usr/local/lib/python3.10/dist-packages/joblib/parallel.py in␣
↪_get_sequential_output(self, iterable)
   1790                 self.n_dispatched_batches += 1
   1791                 self.n_dispatched_tasks += 1
-> 1792                 res = func(*args, **kwargs)
   1793                 self.n_completed_tasks += 1
   1794                 self.print_progress()

/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py in␣
↪__call__(self, *args, **kwargs)
    127             config = {}
    128         with config_context(**config):
--> 129             return self.function(*args, **kwargs)

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py in␣
↪_logistic_regression_path(X, y, pos_class, Cs, fit_intercept, max_iter, tol,␣
↪verbose, solver, coef, class_weight, dual, penalty, intercept_scaling,␣
↪multi_class, random_state, check_input, max_squared_sum, sample_weight,␣
↪l1_ratio, n_threads)
    453                 np.searchsorted(np.array([0, 1, 2, 3]), verbose)
    454             ]
```

```
--> 455             opt_res = optimize.minimize(

    456                 func,
    457                 w0,

/usr/local/lib/python3.10/dist-packages/scipy/optimize/_minimize.py in␣
 ↪minimize(fun, x0, args, method, jac, hess, hessp, bounds, constraints, tol,␣
 ↪callback, options)
    708                                       **options)
    709     elif meth == 'l-bfgs-b':
--> 710         res = _minimize_lbfgsb(fun, x0, args, jac, bounds,

    711                                 callback=callback, **options)
    712     elif meth == 'tnc':

/usr/local/lib/python3.10/dist-packages/scipy/optimize/_lbfgsb_py.py in␣
 ↪_minimize_lbfgsb(fun, x0, args, jac, bounds, disp, maxcor, ftol, gtol, eps,␣
 ↪maxfun, maxiter, iprint, callback, maxls, finite_diff_rel_step,␣
 ↪**unknown_options)
    363             # until the completion of the current minimization iteratio
    364             # Overwrite f and g:
--> 365             f, g = func_and_grad(x)
    366         elif task_str.startswith(b'NEW_X'):
    367             # new iteration

/usr/local/lib/python3.10/dist-packages/scipy/optimize/_differentiable_function
 ↪py in fun_and_grad(self, x)
    283         if not np.array_equal(x, self.x):
    284             self._update_x_impl(x)
--> 285         self._update_fun()
    286         self._update_grad()
    287         return self.f, self.g

/usr/local/lib/python3.10/dist-packages/scipy/optimize/_differentiable_function
 ↪py in _update_fun(self)
    249     def _update_fun(self):
    250         if not self.f_updated:
--> 251             self._update_fun_impl()
    252             self.f_updated = True
    253

/usr/local/lib/python3.10/dist-packages/scipy/optimize/_differentiable_function
 ↪py in update_fun()
    153
    154         def update_fun():
--> 155             self.f = fun_wrapped(self.x)
    156
    157         self._update_fun_impl = update_fun
```

```
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_differentiable_functions.
↪py in fun_wrapped(x)
    135             # Overwriting results in undefined behaviour because
    136             # fun(self.x) will change self.x, with the two no longer␣
↪linked.
--> 137             fx = fun(np.copy(x), *args)
    138             # Make sure the function returns a true scalar
    139             if not np.isscalar(fx):

/usr/local/lib/python3.10/dist-packages/scipy/optimize/_optimize.py in␣
↪__call__(self, x, *args)
     75     def __call__(self, x, *args):
     76         """ returns the function value """
---> 77         self._compute_if_needed(x, *args)
     78         return self._value
     79

/usr/local/lib/python3.10/dist-packages/scipy/optimize/_optimize.py in␣
↪_compute_if_needed(self, x, *args)
     69         if not np.all(x == self.x) or self._value is None or self.jac is␣
↪None:
     70             self.x = np.asarray(x).copy()
---> 71             fg = self.fun(x, *args)
     72             self.jac = fg[1]
     73             self._value = fg[0]

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_linear_loss.py in␣
↪loss_gradient(self, coef, X, y, sample_weight, l2_reg_strength, n_threads,␣
↪raw_prediction)
    274
    275         if raw_prediction is None:
--> 276             weights, intercept, raw_prediction = self.
↪weight_intercept_raw(coef, X)
    277         else:
    278             weights, intercept = self.weight_intercept(coef)

KeyboardInterrupt:
```

[ ]:
```
model_logreg_ = model_logreg.fit(x_train, y_train)
y_pred_ = model_logreg_.predict(x_val)
print(accuracy_score(y_val, y_pred_))
y_pred_ = model_logreg_.predict(x_test)
print(accuracy_score(y_test, y_pred_))
```

```
0.7337662337662337
0.7402597402597403
```

[ ]:
```
model_logreg_b = model_logreg.fit(x_train_b, y_train_b)
y_pred_b = model_logreg_b.predict(x_val_b)
print(accuracy_score(y_val_b, y_pred_b))
y_pred_b = model_logreg_b.predict(x_test_b)
print(accuracy_score(y_test_b, y_pred_b))
```

```
0.71
0.715
```

[ ]:
```
model_logreg_p = model_logreg.fit(x_train_p, y_train_p)
y_pred_p = model_logreg_p.predict(x_val_p)
print(accuracy_score(y_val_p, y_pred_p))
y_pred_p = model_logreg_p.predict(x_test_p)
print(accuracy_score(y_test_p, y_pred_p))
```

```
0.7857142857142857
0.6753246753246753
```

[ ]:
```
model_logreg_b_p = model_logreg.fit(x_train_b_p, y_train_b_p)
y_pred_b_p = model_logreg_p.predict(x_val_b_p)
print(accuracy_score(y_val_b_p, y_pred_b_p))
y_pred_b_p = model_logreg_p.predict(x_test_b_p)
print(accuracy_score(y_test_b_p, y_pred_b_p))
```

```
0.78
0.71
```

[ ]:
```

```