Feature Selection

- There are several benefits of feature selection
- Some are given below:
  - Reduces overfitting: Less redundant data means less opportunity to make decisions based on noise.
  - Improves Accuracy: Less misleading data means modeling accuracy improves.
  - Reduces Training Time: Less data means that algorithms train faster.
- In this notebook, we discuss several feature selection algorithms
  - Algorithm 1: Dropping Constant Features using Variance Threshold Technique
  - Algorithm 2: Feature Selection using Pearson's correlation
  - Algorithm 3: Feature Selection using Information Gain
  - Algorithm 4: Feature Selection using RFECV (Recursive Feature Elimination with Cross Validation)

Algorithm 1 - Dropping Constant Features using Variance Threshold Technique

In [1]:
```
import pandas as pd
# Make a dataframe for the following data

data = pd.DataFrame({"A":[1,2,4,1,2,4],
                     "B":[4,5,6,7,8,9],
                     "C":[0,0,0,0,0,0],
                     "D":[1,1,1,1,1,1]}
                    )
data
```

Out[1]:

	A	B	C	D
0	1	4	0	1
1	2	5	0	1
2	4	6	0	1
3	1	7	0	1
4	2	8	0	1
5	4	9	0	1

Variance Threshold:

- https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html
- class sklearn.feature_selection.VarianceThreshold(threshold=0.0)
- Feature selector that removes all low-variance features.
- This feature selection algorithm looks only at the features (X), not the desired outputs (y), and can thus be used for unsupervised learning.

In [2]:
```
from sklearn.feature_selection import VarianceThreshold
# By default, the threshold = 0 i.e. it will remove the zero-variance threshold
var_thresh = VarianceThreshold() # by default the threshold = 0
var_thresh.fit(data)
```

Out[2]:
```
VarianceThreshold()
```

In [3]:
```
var_thresh.get_support()
```

Out[3]:
```
array([ True,  True, False, False])
```

In [4]:
```
print("All features: ", data.columns)
print("Features Selected: ", data.columns[var_thresh.get_support()])
```

```
All features:  Index(['A', 'B', 'C', 'D'], dtype='object')
Features Selected:  Index(['A', 'B'], dtype='object')
```

In [5]:
```
# the following code drops the columns based on the variance threshold algorithm
selected_columns = data.columns[var_thresh.get_support()]
for cols in data.columns:
    if cols not in selected_columns:
        data.drop(columns = cols, inplace = True)
```

In [6]:
```
data
```

Out[6]:

	A	B
0	1	4
1	2	5
2	4	6
3	1	7
4	2	8
5	4	9

Removing zero variance features using raw python code

In [7]:
```
data = pd.DataFrame({"A":[1,2,4,1,2,4],
                     "B":[4,5,6,7,8,9],
                     "C":[0,0,0,0,0,0],
                     "D":[1,1,1,1,1,1]}
                    )
data
```

Out[7]:

	A	B	C	D
0	1	4	0	1
1	2	5	0	1
2	4	6	0	1
3	1	7	0	1
4	2	8	0	1
5	4	9	0	1

In [8]:
```
selected_features = []
for cols in data.columns:
    if len(data[cols].unique()) > 1:
        selected_features.append(cols)
selected_features
```

Out[8]:
```
['A', 'B']
```

Algorithm 2 - Feature Selection with Pearson's correlation

Idea

- Highly correlated features with the target variable are important features
- High correlation between features, (say over 90% or over 80%) indicate the existence of duplicate features.
- In case of duplicate features, we do not need to take all the features but one one of them would suffice

In [9]:
```
## Let's load the Absenteeism dataset

df = pd.read_excel('absenteeism.xls')
print(df.shape)
df.head()
```

```
(740, 21)
```

Out[9]:

	ID	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time	Age	Work load Average/day	...	Disciplinary failure	Education	Son	Social drinker
0	11	26.0	7.0	3	1	289.0	36.0	13.0	33.0	239554.0	...	0.0	1.0	2.0	1.0
1	36	0.0	7.0	3	1	118.0	13.0	18.0	50.0	239554.0	...	1.0	1.0	1.0	1.0
2	3	23.0	7.0	4	1	179.0	51.0	18.0	38.0	239554.0	...	0.0	1.0	0.0	1.0
3	7	7.0	7.0	5	1	279.0	5.0	14.0	39.0	239554.0	...	0.0	1.0	2.0	1.0
4	11	23.0	7.0	5	1	289.0	36.0	13.0	33.0	239554.0	...	0.0	1.0	2.0	1.0

5 rows × 21 columns

In [10]:
```
df.dropna(inplace = True)
print(df.shape)
```

```
(639, 21)
```

In [11]:
```
df.columns
```

Out[11]:
```
Index(['ID', 'Reason for absence', 'Month of absence', 'Day of the week',
       'Seasons', 'Transportation expense', 'Distance from Residence to Work',
       'Service time', 'Age', 'Work load Average/day ', 'Hit target',
       'Disciplinary failure', 'Education', 'Son', 'Social drinker',
       'Social smoker', 'Pet', 'Weight', 'Height', 'Body mass index',
       'Absenteeism time in hours'],
      dtype='object')
```

In [12]:
```
# Separate the independent (X) and dependent (y) features
y = df['Absenteeism time in hours']
X = df.drop(columns = 'Absenteeism time in hours')
```

In [13]:
```
X.head()
```

Out[13]:

	ID	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time	Age	Work load Average/day	Hit target	Disciplinary failure	Education	Son	Social drinker
0	11	26.0	7.0	3	1	289.0	36.0	13.0	33.0	239554.0	97.0	0.0	1.0	2.0	1.0
1	36	0.0	7.0	3	1	118.0	13.0	18.0	50.0	239554.0	97.0	1.0	1.0	1.0	1.0
2	3	23.0	7.0	4	1	179.0	51.0	18.0	38.0	239554.0	97.0	0.0	1.0	0.0	1.0
3	7	7.0	7.0	5	1	279.0	5.0	14.0	39.0	239554.0	97.0	0.0	1.0	2.0	1.0
4	11	23.0	7.0	5	1	289.0	36.0	13.0	33.0	239554.0	97.0	0.0	1.0	2.0	1.0

In [14]:
```
y.head()
```

Out[14]:
```
0    4.0
1    0.0
2    2.0
3    4.0
4    2.0
Name: Absenteeism time in hours, dtype: float64
```

In [15]:
```
# Let's first drop the feature id as it is a personal identifier
# Let's split the data into train and test set
# Note that correlation will be done only on the training dataset
from sklearn.model_selection import train_test_split
X.drop(columns = "ID", inplace = True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 20)
```

In [16]:
```
# let's import the required libraries
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [17]:
```
X_train.head()
```

Out[17]:

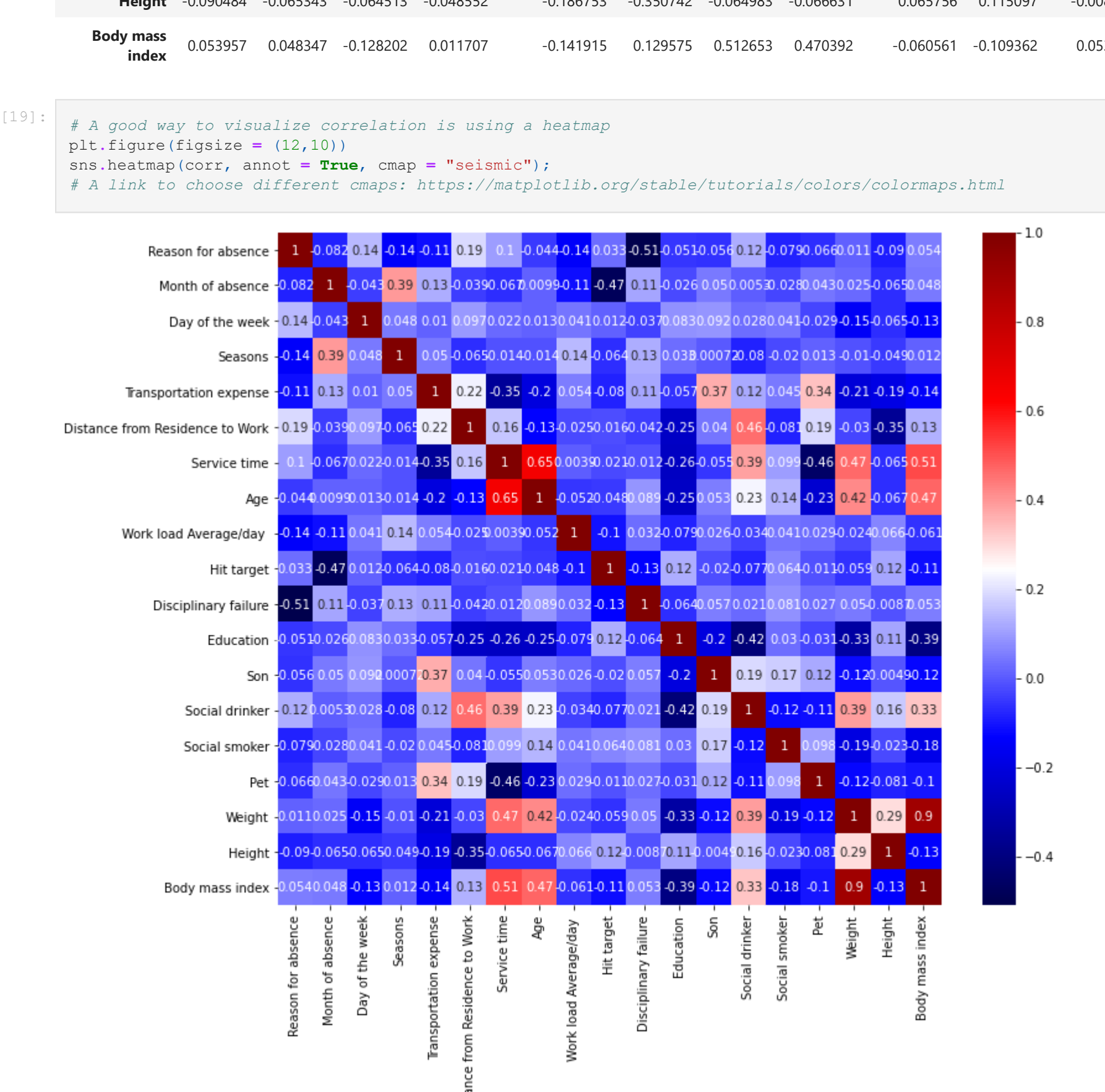
	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time	Age	Work load Average/day	Hit target	Disciplinary failure	Education	Son	Social drinker
645	27.0	3.0	6	2	179.0	51.0	18.0	38.0	222196.0	99.0	0.0	1.0	0.0	1.0
588	27.0	2.0	4	2	179.0	51.0	18.0	38.0	264249.0	97.0	0.0	1.0	0.0	1.0
51	0.0	9.0	2	4	225.0	26.0	9.0	28.0	241476.0	92.0	1.0	1.0	1.0	0.0
54	0.0	9.0	3	4	289.0	36.0	13.0	33.0	241476.0	92.0	1.0	1.0	2.0	1.0
88	23.0	11.0	4	4	225.0	26.0	9.0	28.0	306345.0	93.0	0.0	1.0	1.0	0.0

In [18]:
```
# Let's find the feature correlations

corr = X_train.corr()
corr

# Note that the correlation values lie between -1 and +1
# A correlation value close to -1 indicates a strong negative linear correlation
# A correlation value close to +1 indicates a strong positive linear correlation
# A correlation value close to 0 indicates no linear correlation
```

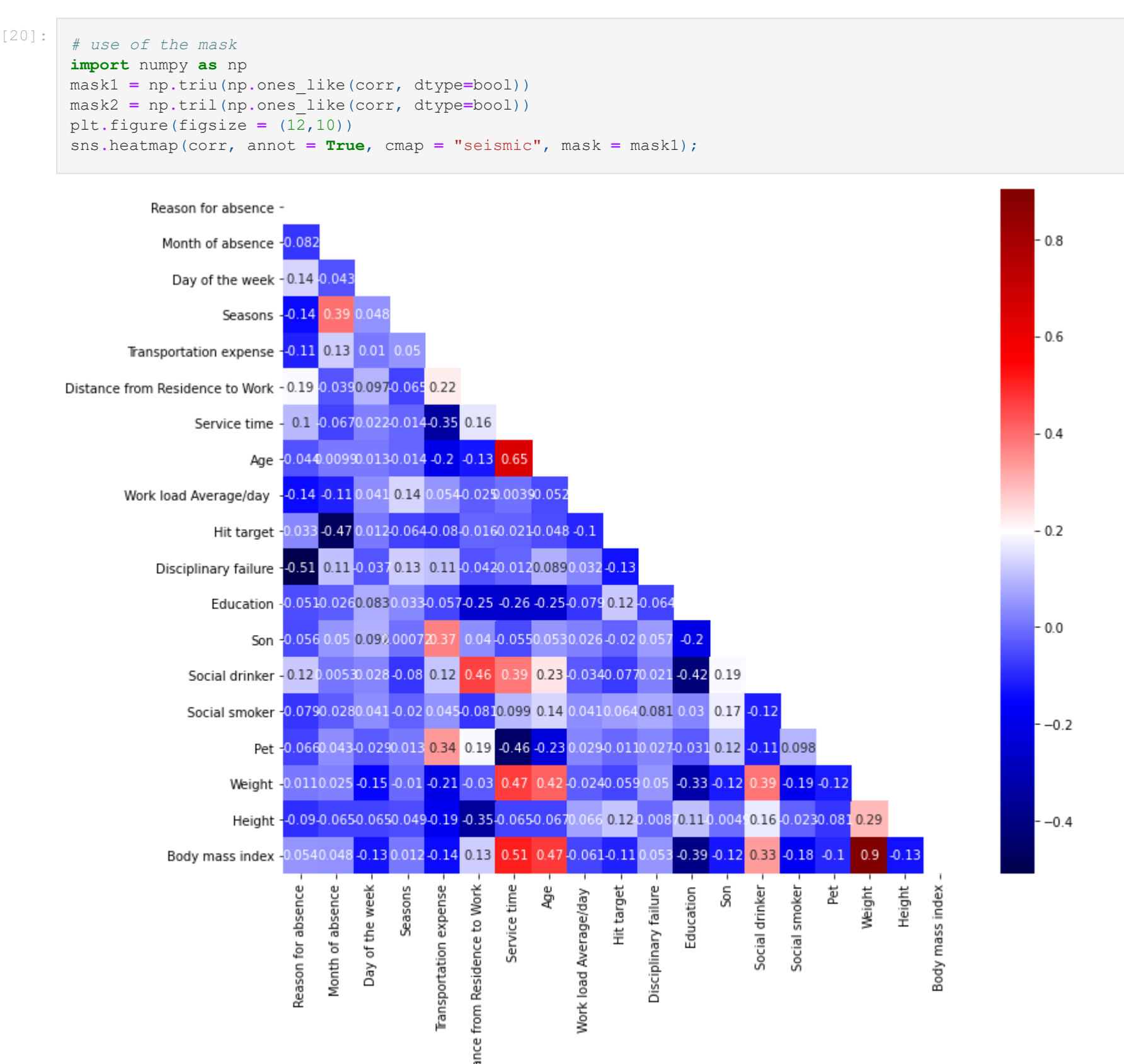
Out[18]:

	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time	Age	Work load Average/day	Hit target	Disciplinary failure
Reason for absence	1.000000	-0.082365	0.139880	-0.141645	-0.108721	0.187522	0.099970	-0.044458	-0.141390	0.032852	-0.509388
Month of absence	-0.082365	1.000000	-0.042561	0.394502	0.126988	-0.039480	-0.066917	0.009941	-0.110607	-0.474779	0.111391
Day of the week	0.139880	-0.042561	1.000000	0.048180	0.010392	0.096749	0.021794	0.013481	0.041432	0.012381	-0.037094
Seasons	-0.141645	0.394502	0.048180	1.000000	0.049554	-0.065337	-0.013556	-0.014083	0.142321	-0.063809	0.126736
Transportation expense	-0.108721	0.126988	0.010392	0.049554	1.000000	0.224753	-0.347612	-0.202000	0.053691	-0.079950	0.111324
Distance from Residence to Work	0.187522	-0.039480	0.096749	-0.065337	0.224753	1.000000	0.162415	-0.127041	-0.024887	-0.016185	-0.042408
Service time	0.099970	-0.066917	0.021794	-0.013556	-0.347612	0.162415	1.000000	0.652352	0.003865	-0.021079	-0.012317
Age	-0.044458	0.009941	0.013481	-0.014083	-0.202000	-0.127041	0.652352	1.000000	-0.051574	-0.048063	0.088810
Work load Average/day	-0.141390	-0.110607	0.041432	0.142321	0.053691	-0.024887	0.003865	-0.051574	1.000000	-0.104236	0.031562
Hit target	0.032852	-0.474779	0.012381	-0.063809	-0.079950	-0.016185	-0.021079	-0.048063	-0.104236	1.000000	-0.128949
Disciplinary failure	-0.509388	0.111391	-0.037094	0.126736	0.111324	-0.042408	-0.012317	0.088810	0.031562	-0.128949	1.000000
Education	-0.050810	-0.025659	0.082978	0.032900	-0.057344	-0.251078	-0.257400	-0.250159	-0.079139	0.123075	-0.064107
Son	-0.056189	0.049621	0.091858	0.000724	0.369093	0.040158	-0.055222	0.052568	0.025612	-0.019617	0.056510
Social drinker	0.121760	0.005340	0.027641	-0.080454	0.117376	0.458648	0.385568	0.231321	-0.033912	-0.077325	0.021469
Social smoker	-0.079242	-0.028218	0.040792	-0.020047	0.045165	-0.080751	0.099249	0.137464	0.040739	0.064472	0.081115
Pet	-0.066426	0.042912	-0.029048	0.012827	0.338960	0.190246	-0.455354	-0.227388	0.028977	-0.011375	0.027295
Weight	0.010671	0.024668	-0.145997	-0.010173	-0.209951	-0.029693	0.465684	0.418176	-0.024456	-0.059414	0.049689
Height	-0.090484	-0.065343	-0.064513	-0.048552	-0.186753	-0.350742	-0.064983	-0.066631	0.065756	0.115097	-0.008680
Body mass index	0.053957	0.048347	-0.128202	0.011707	-0.141915	0.129575	0.512653	0.470392	-0.060561	-0.109362	0.053245

In [19]:
```
# A good way to visualize correlation is using a heatmap
plt.figure(figsize = (12,10))
sns.heatmap(corr, annot = True, cmap = "seismic");
# A link to choose different cmaps: https://matplotlib.org/stable/tutorials/colors/colormaps.html
```



In [20]:
```
# use of the mask
import numpy as np
mask1 = np.triu(np.ones_like(corr, dtype=bool))
mask2 = np.tril(np.ones_like(corr, dtype=bool))
plt.figure(figsize = (12,10))
sns.heatmap(corr, annot = True, cmap = "seismic", mask = mask1);
```



In [21]:
```
# with the following function we can select highly correlated features
# it will remove the first feature that is highly correlated with another feature

def correlation(dataset, threshold):
    col_corr = set() # Set of all the names of the redundant columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if(abs(corr_matrix.iloc[i, j])) > threshold:
                colname = corr_matrix.columns[i]
                col_corr.add(colname)
    return col_corr
```

In [22]:
```
corr_features = correlation(X_train, 0.9) # 85% is a good value of the threshold
print(len(corr_features))
print(corr_features)
```

```
1
{'Body mass index'}
```

Algorithm 3 - Feature Selection using Information Gain

In [23]:
```
# Let's read a dataset [PIMA Indians Diabetes Dataset]
df = pd.read_csv('diabetes.csv')
print(df.shape)
df.head()
```

```
(768, 9)
```

Out[23]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [24]:
```
df = df.dropna()
print("Shape: ", df.shape)
df["Outcome"].value_counts()
```

```
Shape:  (768, 9)
```

Out[24]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

In [25]:
```
# Let's separate the independent (X) and dependent (y) variables
y = df['Outcome']
X = df.drop(columns = 'Outcome')
X.head()
```

Out[25]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

In [26]:
```
# Split the dataset into train and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2 )
```

In [27]:
```
# URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html
# Estimate mutual information for a discrete target variable.
# Mutual information (MI) between two random variables is a non-negative value, which measures the dependency b
# It is equal to zero if and only if two random variables are independent, and higher values mean higher depend
from sklearn.feature_selection import mutual_info_classif

mutual_info = mutual_info_classif(X_train, y_train, random_state = 20)
mutual_info
```

Out[27]:
```
array([0.04738501, 0.12236251, 0.01140381, 0.01226112, 0.04213022,
       0.0729025 , 0.00461064, 0.07182349])
```

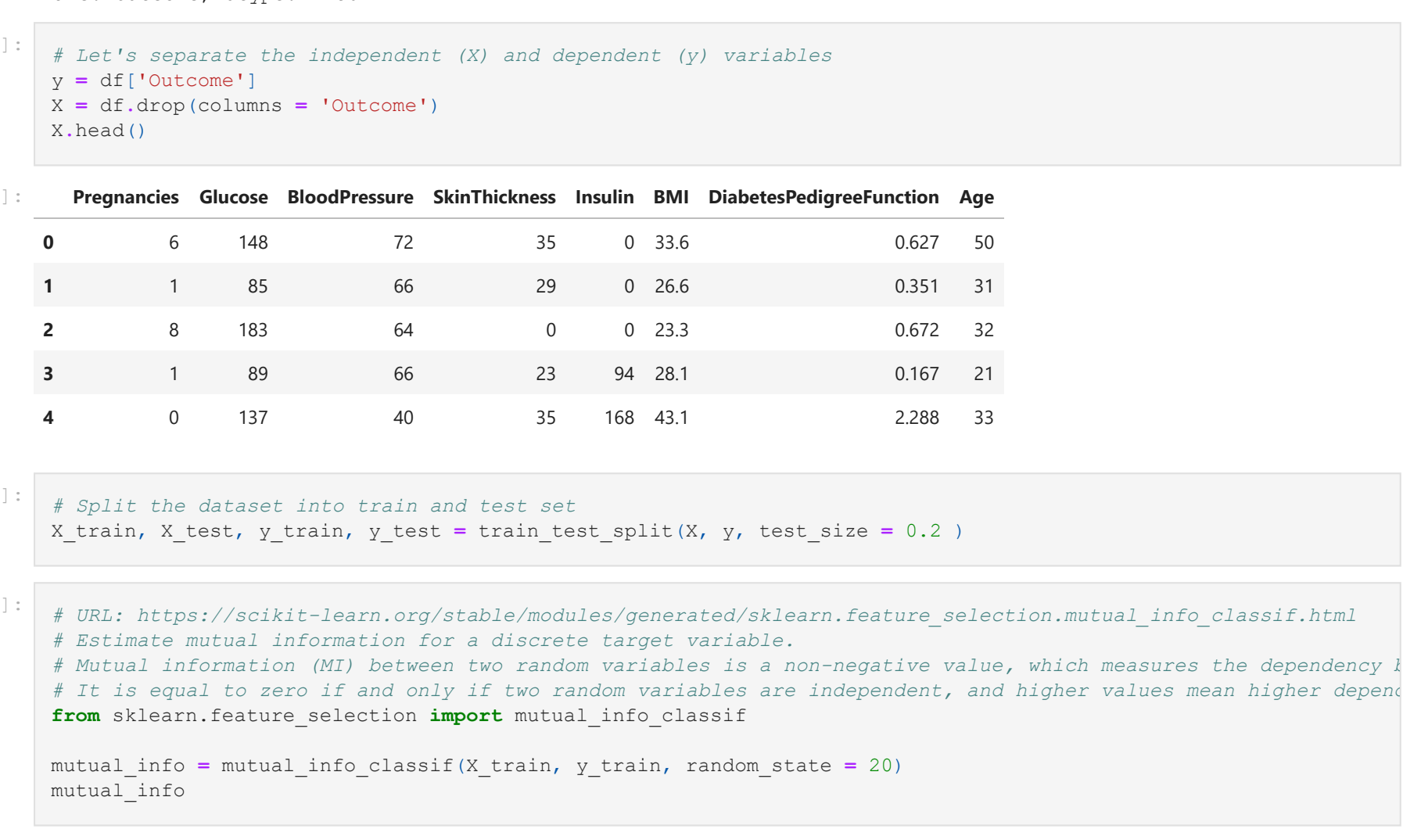
In [28]:
```
print(len(mutual_info))
print(X_train.shape)
```

```
8
(614, 8)
```

In [29]:
```
mutual_info = pd.Series(mutual_info)
mutual_info.index = X_train.columns
mutual_info.sort_values(ascending = False)
```

Out[29]:
```
Glucose                     0.122363
BMI                         0.072903
Age                         0.071823
Pregnancies                 0.047385
Insulin                     0.042130
SkinThickness               0.012261
BloodPressure               0.011404
DiabetesPedigreeFunction    0.004611
dtype: float64
```

In [30]:
```
plt.figure(figsize = (20, 8))
mutual_info.sort_values(ascending = False).plot.bar();
```



Algorithm 4 - Feature Selection using RFECV (Recursive Feature Elimination with Cross Validation)

- URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html
- URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFECV.html
- Feature ranking with recursive feature elimination.

RFE: Given an external estimator that assigns weights to features, the goal of recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features and the importance of each feature is obtained either through any specific attribute or callable. Then, the least important features are pruned from current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached. RFECV: Feature ranking with recursive feature elimination and cross-validated selection of the best number of features.

In [31]:
```
X_train
```

Out[31]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
607	1	92	62	25	41	19.5	0.482	25
58	0	146	82	0	0	40.5	1.781	44
15	7	100	0	0	0	30.0	0.484	32
711	5	126	78	27	22	29.6	0.439	40
756	7	137	90	41	0	32.0	0.391	39
...	...	...	...	...	...	...	...	...
726	1	116	78	29	180	36.1	0.496	25
565	2	95	54	14	88	26.1	0.748	22
290	0	78	88	29	40	36.9	0.434	21
291	0	107	62	30	74	36.6	0.757	25
75	1	0	48	20	0	24.7	0.140	22

614 rows × 8 columns

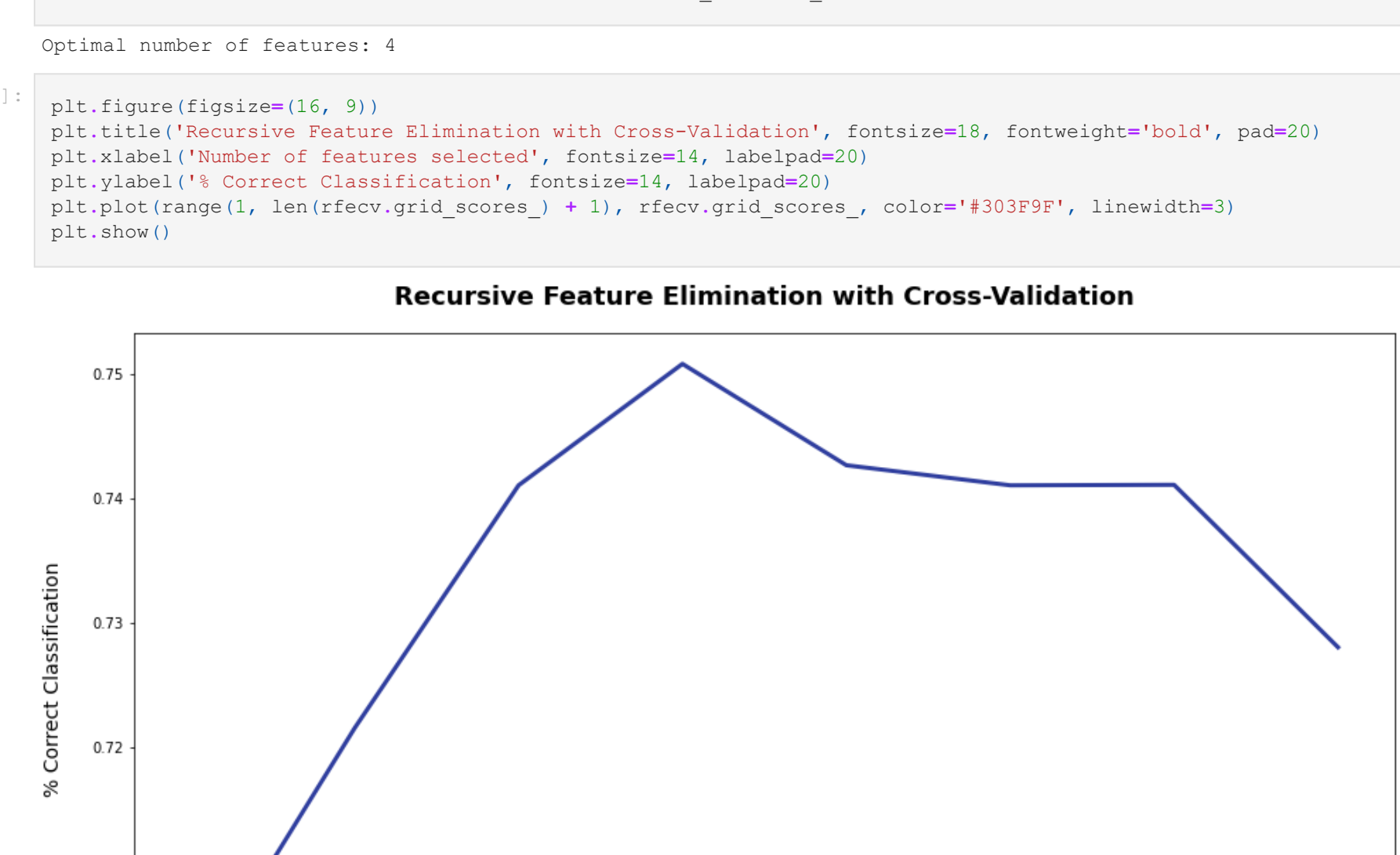
In [32]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFECV
from sklearn.model_selection import StratifiedKFold
import warnings
warnings.filterwarnings('ignore')

rfc = RandomForestClassifier(random_state=101)
rfecv = RFECV(estimator=rfc, step=1, cv=StratifiedKFold(5), scoring='accuracy')
rfecv.fit(X_train, y_train)
```

Out[32]:
```
RFECV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
      estimator=RandomForestClassifier(random_state=101), scoring='accuracy')
```

In [33]:
```
print('Optimal number of features: {}'.format(rfecv.n_features_))
```

```
Optimal number of features: 4
```

In [34]:
```
plt.figure(figsize=(16, 9))
plt.title('Recursive Feature Elimination with Cross-Validation', fontsize=18, fontweight='bold', pad=20)
plt.xlabel('Number of features selected', fontsize=14, labelpad=20)
plt.ylabel('% Correct Classification', fontsize=14, labelpad=20)
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_, color='#303F9F', linewidth=3)
plt.show()
```



In [35]:
```
print(np.where(rfecv.support_ == False)[0])
```

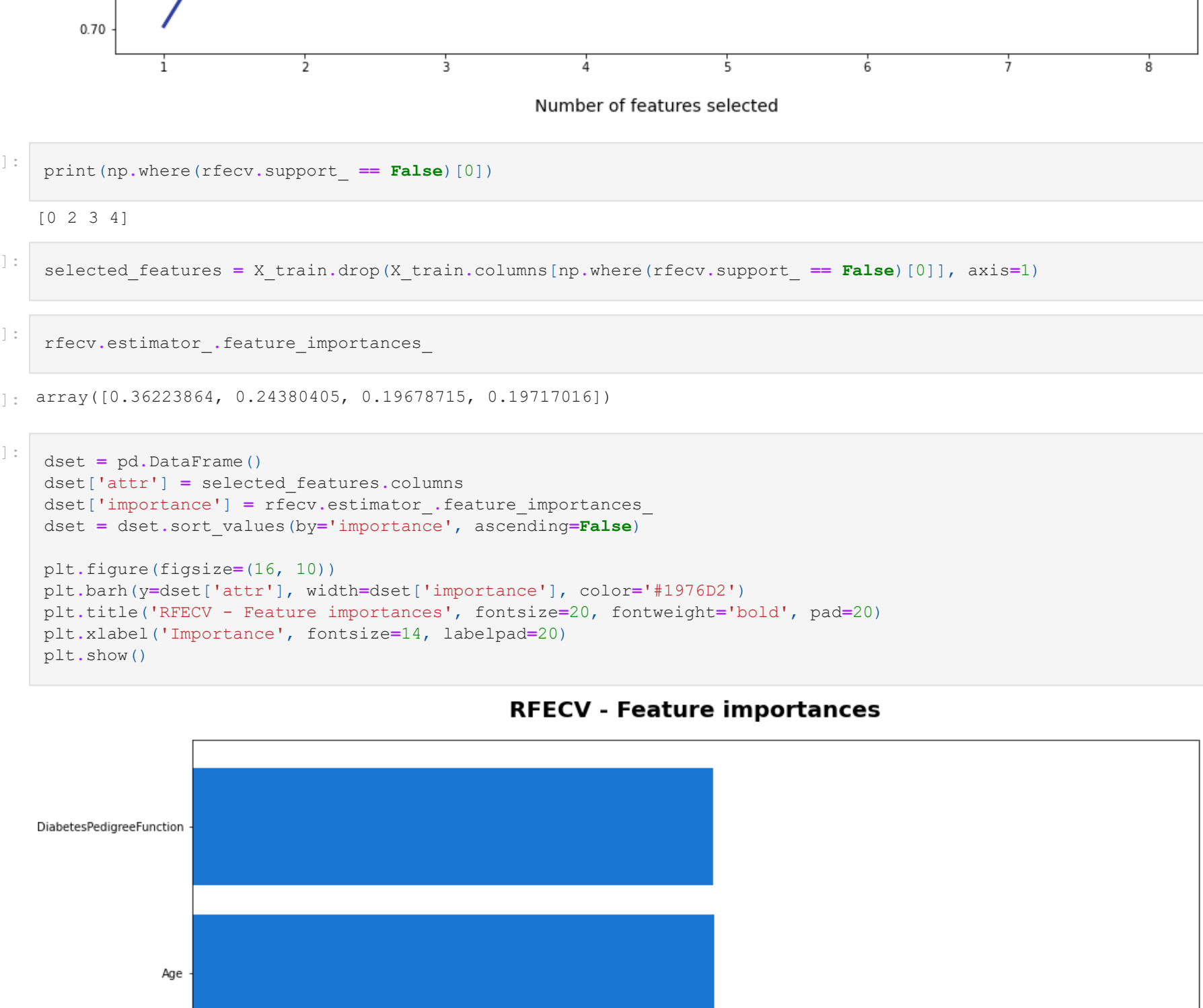
```
[0 2 3 4]
```

In [36]:
```
selected_features = X_train.drop(X_train.columns[np.where(rfecv.support_ == False)[0]], axis=1)
```

In [37]:
```
rfecv.estimator_.feature_importances_
```

Out[37]:
```
array([0.36223864, 0.24380405, 0.19678715, 0.19717016])
```

In [38]:
```
dset = pd.DataFrame()
dset['attr'] = selected_features.columns
dset['importance'] = rfecv.estimator_.feature_importances_
dset = dset.sort_values(by='importance', ascending=False)

plt.figure(figsize=(16, 10))
plt.barh(y=dset['attr'], width=dset['importance'], color='#1976D2')
plt.title('RFECV - Feature importances', fontsize=20, fontweight='bold', pad=20)
plt.xlabel('Importance', fontsize=14, labelpad=20)
plt.show()
```



In [39]:
```
dset
```

Out[39]:

	attr	importance
0	Glucose	0.362239
1	BMI	0.243804
3	Age	0.197170
2	DiabetesPedigreeFunction	0.196787

In [ ]: