Import Libraries we want to use

[2]:
```
import pandas as pd
#For virtualization
import seaborn as sns
import matplotlib.pyplot as plt
```

Loading Datasets

[4]:
```
# Load the baby weights dataset
dataset = pd.read_csv('baby-weights-dataset.csv')
dataset
```

[4]:

	ID	SEX	MARITAL	FAGE	GAINED	VISITS	MAGE	FEDUC	MEDUC	TOTALP	...	HYPERCH	HYPERPR	ECLAMP	CERVIX	PINFANT	PRETERM	RENAL	RHS
0	2001	2	1	33	26.0	10	34	12.0	4	2	...	0	0	0	0	0	0	0	
1	2002	2	2	19	40.0	10	18	11.0	12	1	...	0	0	0	0	0	0	0	
2	2003	2	1	33	16.0	14	31	16.0	16	2	...	0	0	0	0	0	0	0	
3	2004	1	1	25	40.0	15	28	12.0	12	3	...	0	0	0	0	0	0	0	
4	2005	1	2	21	60.0	13	20	12.0	14	2	...	0	1	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
101395	103396	1	2	36	0.0	9	34	3.0	12	4	...	0	0	0	0	0	0	0	
101396	103397	2	2	21	39.0	11	19	12.0	9	2	...	0	0	0	0	0	0	0	
101397	103398	2	1	27	37.0	15	22	12.0	12	2	...	0	0	0	0	0	0	0	
101398	103399	1	1	27	26.0	12	24	12.0	14	1	...	0	0	0	0	0	0	0	
101399	103400	1	2	20	31.0	15	17	12.0	11	1	...	0	0	0	0	0	0	0	

101400 rows × 37 columns

Data Preprocessing

Data Exploration

describing columns in dataset .. provides a summary of each feature, including mean, standard deviation, and quartiles.

[7]:
```
dataset.describe()
```

[7]:

	ID	SEX	MARITAL	FAGE	GAINED	VISITS	MAGE	FEDUC	MEDUC	TOTALP	...	HY
count	101400.000000	101400.000000	101400.000000	101400.000000	101399.000000	101400.000000	101400.000000	101399.000000	101400.000000	101400.000000	...	101400
mean	52700.500000	1.485671	1.303817	30.174477	30.283040	12.436943	27.736312	12.926883	13.256489	2.378462	...	0
std	29271.802985	0.500349	0.459907	6.775576	13.615468	3.728901	5.957369	2.926582	2.932693	1.490272	...	0
min	2001.000000	1.000000	1.000000	14.000000	0.000000	0.000000	11.000000	0.000000	0.000000	1.000000	...	0
25%	27350.750000	1.000000	1.000000	25.000000	21.000000	10.000000	23.000000	12.000000	12.000000	1.000000	...	0
50%	52700.500000	1.000000	1.000000	30.000000	30.000000	12.000000	28.000000	12.000000	13.000000	2.000000	...	0
75%	78050.250000	2.000000	2.000000	35.000000	39.000000	15.000000	32.000000	16.000000	16.000000	3.000000	...	0
max	103400.000000	9.000000	2.000000	74.000000	98.000000	49.000000	53.000000	17.000000	17.000000	20.000000	...	1

8 rows × 35 columns

dataset size : number of columns , rows

[9]:
```
dataset.shape
```

[9]:
```
(101400, 37)
```

dataset information

[11]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101400 entries, 0 to 101399
Data columns (total 37 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   ID         101400 non-null  int64
 1   SEX        101400 non-null  int64
 2   MARITAL    101400 non-null  int64
 3   FAGE       101400 non-null  int64
 4   GAINED     101399 non-null  float64
 5   VISITS     101400 non-null  int64
 6   MAGE       101400 non-null  int64
 7   FEDUC      101399 non-null  float64
 8   MEDUC      101400 non-null  int64
 9   TOTALP     101400 non-null  int64
 10  BDEAD      101400 non-null  int64
 11  TERMS      101400 non-null  int64
 12  LOUTCOME   101400 non-null  int64
 13  WEEKS      101399 non-null  float64
 14  RACEMOM    101400 non-null  int64
 15  RACEDAD    101400 non-null  int64
 16  HISPMOM    101400 non-null  object
 17  HISPDAD    101400 non-null  object
 18  CIGNUM     101399 non-null  float64
 19  DRINKNUM   101400 non-null  int64
 20  ANEMIA     101400 non-null  int64
 21  CARDIAC    101400 non-null  int64
 22  ACLUNG     101400 non-null  int64
 23  DIABETES   101400 non-null  int64
 24  HERPES     101400 non-null  int64
 25  HYDRAM     101399 non-null  float64
 26  HEMOGLOB   101400 non-null  int64
 27  HYPERCH    101400 non-null  int64
 28  HYPERPR    101400 non-null  int64
 29  ECLAMP     101400 non-null  int64
 30  CERVIX     101400 non-null  int64
 31  PINFANT    101400 non-null  int64
 32  PRETERM    101400 non-null  int64
 33  RENAL      101400 non-null  int64
 34  RHSEN      101400 non-null  int64
 35  URINE      101400 non-null  int64
 36  BWEIGHT    101400 non-null  float64
dtypes: float64(6), int64(29), object(2)
memory usage: 28.6+ MB
```

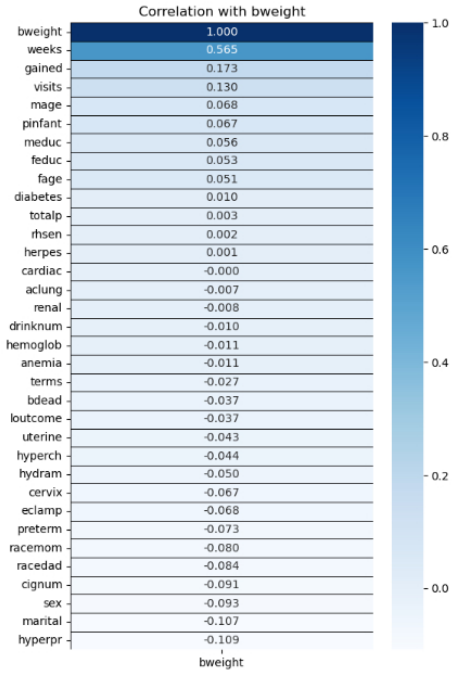
[12]:
```
# make column name lowercase
dataset.columns = dataset.columns.str.lower()
dataset.columns
```

[12]:
```
Index(['id', 'sex', 'marital', 'fage', 'gained', 'visits', 'mage', 'feduc',
       'meduc', 'totalp', 'bdead', 'terms', 'loutcome', 'weeks', 'racemom',
       'racedad', 'hispmom', 'hispdad', 'cignum', 'drinknum', 'anemia',
       'cardiac', 'aclung', 'diabetes', 'herpes', 'hydram', 'hemoglob',
       'hyperch', 'hyperpr', 'eclamp', 'cervix', 'pinfant', 'preterm', 'renal',
       'rhsen', 'uterine', 'bweight'],
      dtype='object')
```

[13]:
```
# Drop the id column
```

```
dataset.drop(columns=['id'], inplace=True) # can use index nested
```

[14]:
```
# numeric columns
numeric_columns = dataset.select_dtypes(include=['number']).columns

# calc correlation
correlation_matrix = dataset[numeric_columns].corr()
correlation_with_bweight = correlation_matrix[['bweight']].sort_values(by='bweight', ascending=False)

# heatmap
plt.figure(figsize=(6, 10))
sns.heatmap(correlation_with_bweight, annot=True, cmap='Blues', linewidths=0.5, linecolor='black', fmt='.3f')
plt.title('Correlation with bweight')
plt.yticks(rotation=0)

# Save the heatmap
plt.savefig('heatmap_bweight.png')
```



Checking the distribution of data in categorical different classes

[16]:
```
# Check the distribution of the 'sex' column
dataset['sex'].value_counts()
```

[16]:
```
sex
1    52160
2    49239
9        1
Name: count, dtype: int64
```

we notice outlier in value 9

[18]:
```
# Check the distribution of the 'marital' column
dataset['marital'].value_counts()
```

[18]:
```
marital
1    70593
2    30807
Name: count, dtype: int64
```

[19]:
```
# Check the distribution of the 'racemom' column
dataset['racemom'].value_counts()
```

[19]:
```
racemom
1    78343
2    18015
8     3269
3     1196
0      224
7      159
4      132
5       42
6       20
Name: count, dtype: int64
```

[20]:
```
# Value counts categorical columns
for col in ['racedad', 'hispmom', 'hispdad']:
    print(dataset[col].value_counts())
    print('-'*20)
```

```
racedad
1    76285
2    20500
8     2915
3     1190
0      205
4      123
7       97
9       50
6       18
5       17
Name: count, dtype: int64
--------------------
hispmom
N    85040
M    11290
S     3757
P      901
C      185
O      135
U       92
Name: count, dtype: int64
--------------------
hispdad
N    84301
M    12039
S     3771
P      884
C      186
O      118
U      101
Name: count, dtype: int64
--------------------
```

Checking For missing Values

[22]:
```
dataset.isnull().sum() #sum of null values
```

[22]:
```
sex        0
marital    0
fage       0
gained     1
```

```
visits      0
mage        0
feduc       1
meduc       0
totalp      0
bdead       0
terms       0
loutcome    0
weeks       1
racemom     0
racedad     0
hispmom     0
hispdad     0
cignum      1
drinknum    0
anemia      0
cardiac     0
aclung      0
diabetes    0
herpes      0
hydram      1
hemoglob    0
hyperch     0
hyperpr     0
eclamp      0
cervix      0
pinfant     0
preterm     0
renal       0
rhsen       0
uterine     0
bweight     0
dtype: int64
```

[23]:
```
dataset.isnull().sum().sum() # num of rows has null values
```

[23]:
```
5
```

Checking for duplication

[25]:
```
dataset.duplicated().sum() # sum of duplicated values
```

[25]:
```
2
```

[26]:
```
dataset[dataset.duplicated()]
```

[26]:

	sex	marital	fage	gained	visits	mage	feduc	meduc	totalp	bdead	...	hyperch	hyperpr	eclamp	cervix	pinfant	preterm	renal	rhsen	uterine	bweight
45040	2	1	20	25.0	12	22	12.0	14	2	0	...	0	0	0	0	0	0	0	0	0	1.4375
54306	1	2	28	11.0	5	26	10.0	10	6	0	...	0	0	0	0	0	0	0	0	0	3.6250

2 rows × 36 columns

Handling missing values and duplicates ensures that the dataset is free from errors and inconsistencies, which is crucial for accurate modeling.

Data Cleaning

Handel Missing values

[28]:
```
# Drop rows with any missing values
dataset.dropna(inplace=True)
dataset.isnull().sum() # should be no missing values
```

[28]:
```
sex         0
marital     0
fage        0
gained      0
visits      0
mage        0
feduc       0
meduc       0
totalp      0
bdead       0
terms       0
loutcome    0
weeks       0
racemom     0
racedad     0
hispmom     0
hispdad     0
cignum      0
drinknum    0
anemia      0
cardiac     0
aclung      0
diabetes    0
herpes      0
hydram      0
hemoglob    0
hyperch     0
hyperpr     0
eclamp      0
cervix      0
pinfant     0
preterm     0
renal       0
rhsen       0
uterine     0
bweight     0
dtype: int64
```

Handel duplicated values with drop/delete

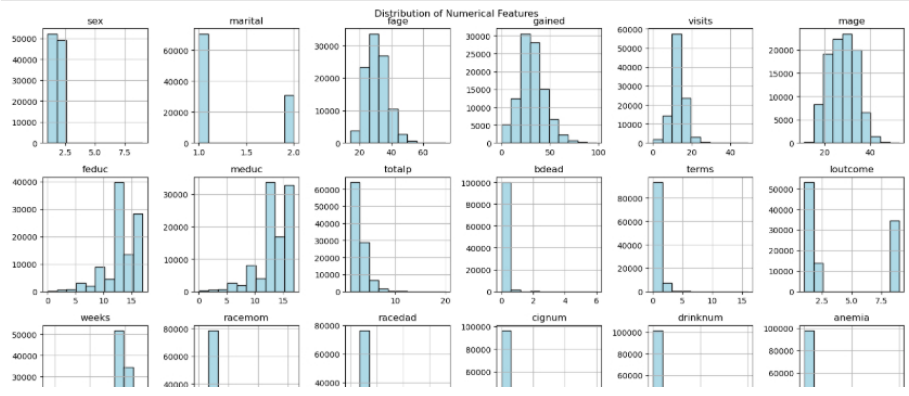
[30]:
```
print(len(dataset))
dataset = dataset.drop_duplicates(keep='first')
print(len(dataset))
```

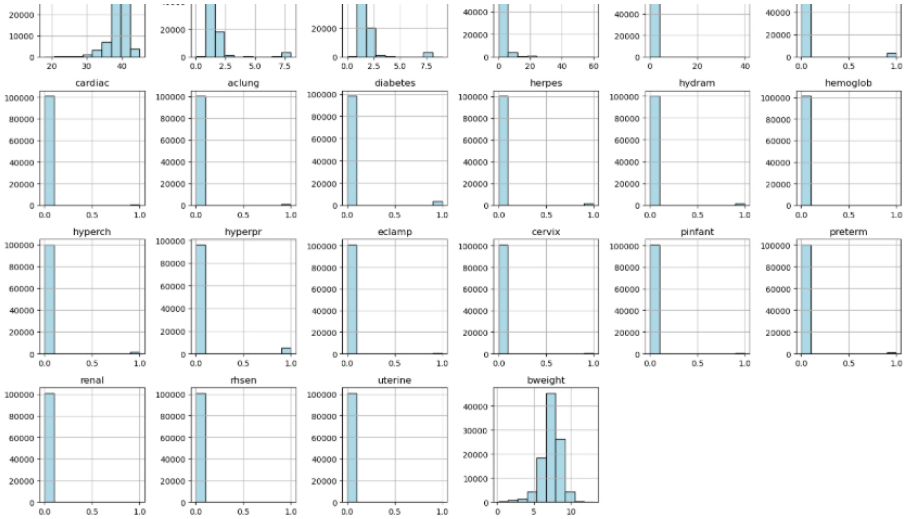
```
101395
101393
```

Data Visualization

show distribution of data

[33]:
```
# Plot histograms for numerical features
numeric_columns = dataset.select_dtypes(include=['float64', 'int64']).columns
dataset[numeric_columns].hist(bins=10, figsize=(16, 16), color='lightblue', edgecolor='black')
plt.suptitle('Distribution of Numerical Features')
plt.tight_layout()
plt.savefig('hist_befor_handel_outlier.png')
```





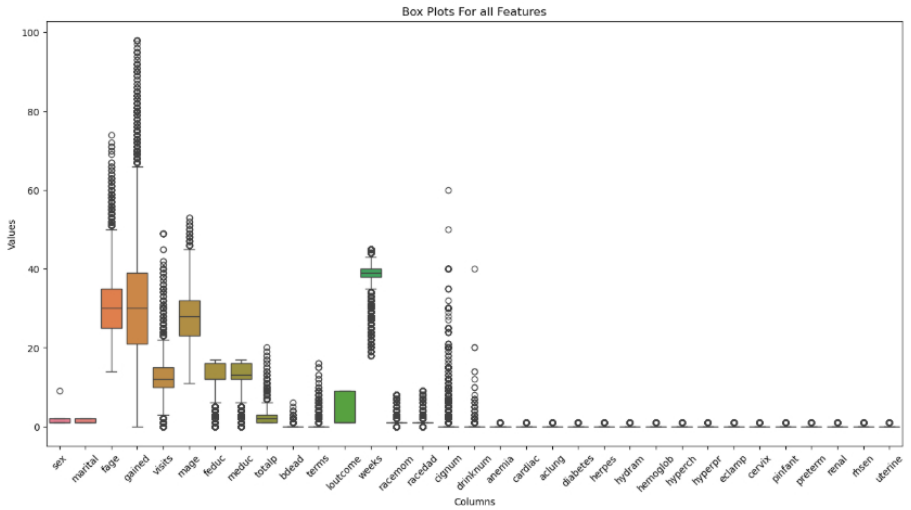
notic outlier & unnormal distribution on some attrebutes like totalap , fage , visits , sex and so on , each histogram explain data destribution and if exsist very large or very small data

Handel outlier values

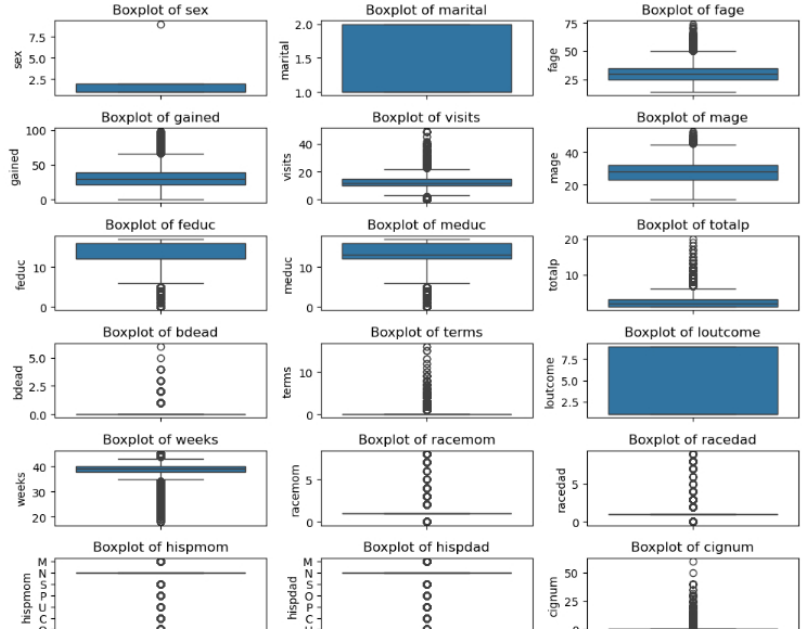
use sns to make boxplot and show data outliers

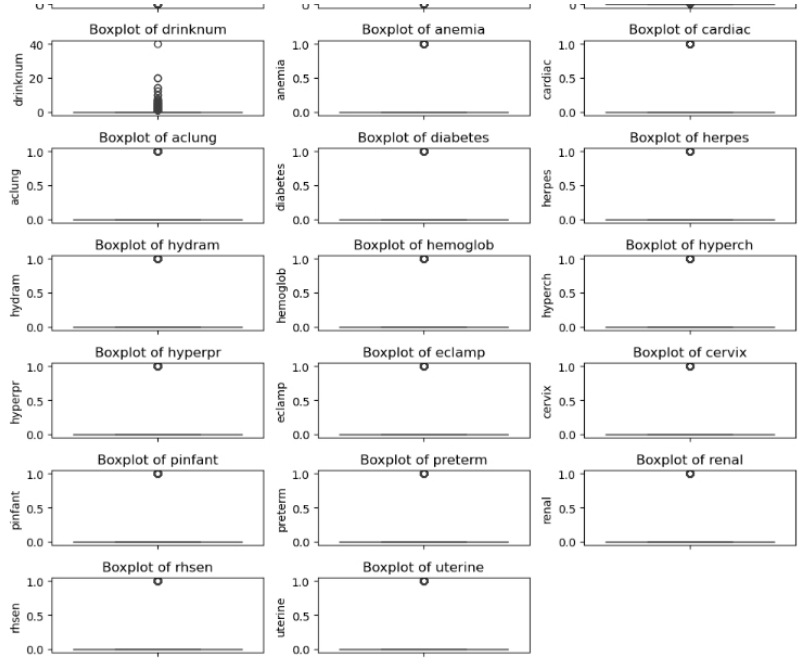
[37]:
```
# boxplot for all columns in the dataset
plt.figure(figsize=(16, 8))
sns.boxplot(data=dataset.iloc[:, :-1]) # except last/object column
plt.title('Box Plots For all Features')
plt.xticks(rotation=45)
plt.ylabel('Values')
plt.xlabel('Columns')

# plt.savefig('single_boxplot.png')
```

[37]:
```
Text(0.5, 0, 'Columns')
```



[38]:
```
plt.figure(figsize=(10, 16))

# boxplots for each column
for i, column in enumerate(dataset.columns[:-1], 1):  # except the last/object column
    plt.subplot(12, 3, i)  # in 12 rows , 3 columns
    sns.boxplot(data=dataset, y=column)
    plt.title(f'Boxplot of {column}')
plt.tight_layout()  # spacing between subplots

# plt.savefig('individual_boxplots.png')
```





[39]:
```
# handel sex column has outlier with value 9
print("len of dataset befor drop sex column outlier : ", len(dataset))
dataset = dataset[dataset['sex'] != 9]
print("len of dataset after drop sex column outlier : ", len(dataset))
```

```
len of dataset befor drop sex column outlier :  101393
len of dataset after drop sex column outlier :  101392
```

[40]:
```
def handle_binary_columns(dataset, binary_columns):
    # hande binary columns -  only rows with 0 or 1 values
    for col in binary_columns:
        dataset = dataset[dataset[col].isin([0, 1])]

    return dataset
```

[41]:
```
print(len(dataset))
dataset = handle_binary_columns(dataset , ['anemia', 'cardiac', 'aclung', 'diabetes', 'herpes']) # verify but boxplot display no outlier
print(len(dataset))
```

```
101392
101392
```

[42]:
```
def remove_outliers(dataset, columns):
    for col in columns:
        if dataset[col].dtype != 'object':
            q1_value = dataset[col].quantile(0.25)
            q3_value = dataset[col].quantile(0.75)
            IQR = q3_value - q1_value
            max_value = q3_value + (1.5 * IQR)
            min_value = q1_value - (1.5 * IQR)
            dataset = dataset[(dataset[col] >= min_value) & (dataset[col] <= max_value)]
    return dataset
```

[43]:
```
print(len(dataset))
dataset = remove_outliers(dataset , ['fage', 'gained', 'visits', 'mage', 'feduc', 'meduc', 'totalp'])
print(len(dataset))
```

```
101392
94054
```

[44]:
```
dataset[['drinknum', 'cignum', 'hispmom', 'hispdad', 'racemom', 'racedad', 'weeks']].describe()
```

[44]:

	drinknum	cignum	racemom	racedad	weeks
count	94054.000000	94054.000000	94054.000000	94054.000000	94054.000000
mean	0.004827	0.708433	1.441725	1.441108	38.766794
std	0.161786	2.923391	1.302274	1.246343	2.457950
min	0.000000	0.000000	0.000000	0.000000	18.000000
25%	0.000000	0.000000	1.000000	1.000000	38.000000
50%	0.000000	0.000000	1.000000	1.000000	39.000000
75%	0.000000	0.000000	1.000000	1.000000	40.000000
max	20.000000	60.000000	8.000000	9.000000	45.000000

Handel outlier on 'drinknum', 'cignum' with mean value

[46]:
```
# IQR for drinknum
Q1_drinknum = dataset['drinknum'].quantile(0.25)
Q3_drinknum = dataset['drinknum'].quantile(0.75)
IQR_drinknum = Q3_drinknum - Q1_drinknum
min_drinknum = Q1_drinknum - 1.5 * IQR_drinknum
max_drinknum = Q3_drinknum + 1.5 * IQR_drinknum

# mean value of drinknum
mean_drinknum = dataset['drinknum'].mean()

# mean value
dataset.loc[(dataset['drinknum'] < min_drinknum) | (dataset['drinknum'] > max_drinknum), 'drinknum'] = mean_drinknum

dataset['drinknum'].describe()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_14296\302503016.py:12: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future version of pandas. Value '0.004827014268390045' has dtype incompatible with int64, please explicitly cast to a compatible dtype first.
  dataset.loc[(dataset['drinknum'] < min_drinknum) | (dataset['drinknum'] > max_drinknum), 'drinknum'] = mean_drinknum
```

[46]:
```
count    94054.000000
mean         0.000011
std          0.000228
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          0.004827
Name: drinknum, dtype: float64
```

[47]:
```
# IQR for cignum
Q1_cignum = dataset['cignum'].quantile(0.25)
Q3_cignum = dataset['cignum'].quantile(0.75)
IQR_cignum = Q3_cignum - Q1_cignum
max_cignum = Q1_cignum - 1.5 * IQR_cignum
min_cignum = Q3_cignum + 1.5 * IQR_cignum

# mean value of cignum
mean_cignum = dataset['cignum'].mean()

# mean value nested of outlier
dataset.loc[(dataset['cignum'] < max_cignum) | (dataset['cignum'] > min_cignum), 'cignum'] = mean_cignum
```

```
# Verify the changes
dataset['cignum'].describe()
```

[47]:
```
count    94054.000000
mean         0.056657
std          0.192167
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          0.708433
Name: cignum, dtype: float64
```

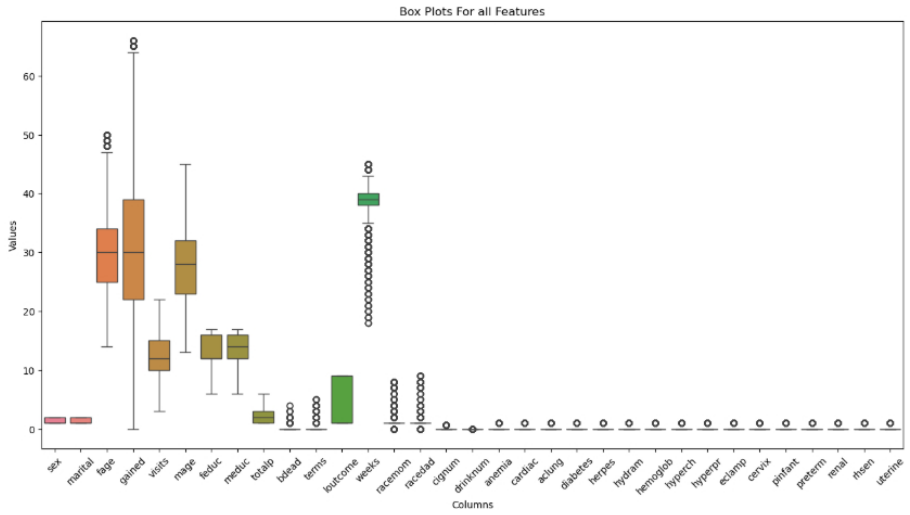
[48]:
```
dataset[['drinknum', 'cignum', 'hispmom', 'hispdad', 'racemom', 'racedad', 'weeks']].describe()
```

[48]:

	drinknum	cignum	racemom	racedad	weeks
count	94054.000000	94054.000000	94054.000000	94054.000000	94054.000000
mean	0.000011	0.056657	1.441725	1.441108	38.766794
std	0.000228	0.192167	1.302274	1.246343	2.457950
min	0.000000	0.000000	0.000000	0.000000	18.000000
25%	0.000000	0.000000	1.000000	1.000000	38.000000
50%	0.000000	0.000000	1.000000	1.000000	39.000000
75%	0.000000	0.000000	1.000000	1.000000	40.000000
max	0.004827	0.708433	8.000000	9.000000	45.000000

[49]:
```
# boxplot for all columns in the dataset
plt.figure(figsize=(16, 8))
sns.boxplot(data=dataset.iloc[:, :-1]) # except last/object column
plt.title('Box Plots For all Features')
plt.xticks(rotation=45)
plt.ylabel('Values')
plt.xlabel('Columns')

# plt.savefig('single_boxplot_after_drop_outlier.png')
```

[49]:
```
Text(0.5, 0, 'Columns')
```



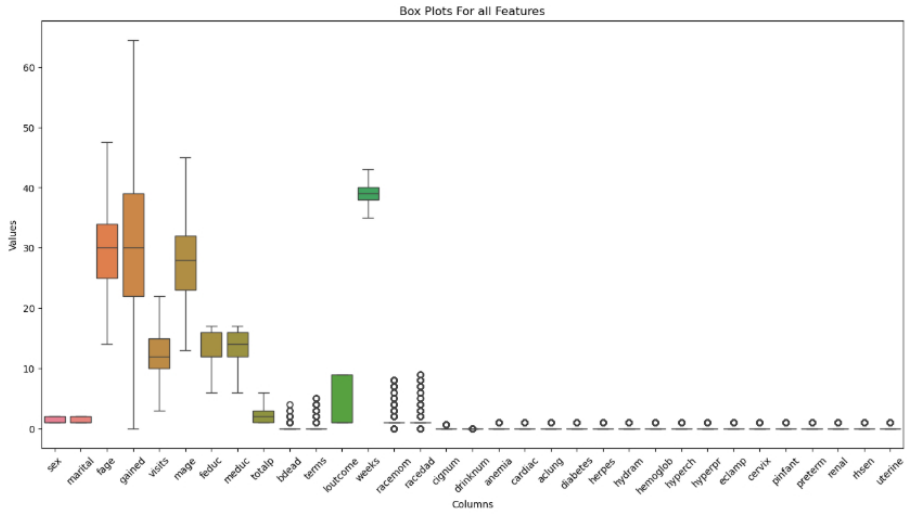
[50]:
```
# other function to handel outlier : Clip
def clip_outliers(dataset , columns):
  for col in columns:
    if dataset[col].dtype != 'object':
      q1_value = dataset[col].quantile(0.25)
      q3_value = dataset[col].quantile(0.75)
      IQR = q3_value - q1_value
      max = q3_value + (1.5 * IQR)
      min = q1_value - (1.5 * IQR)
      dataset[col] = dataset[col].clip(lower=min, upper=max)
  return dataset
```

[51]:
```
print(len(dataset))
dataset = clip_outliers(dataset , ['fage','gained','weeks'])
print(len(dataset))
```

```
94054
94054
```

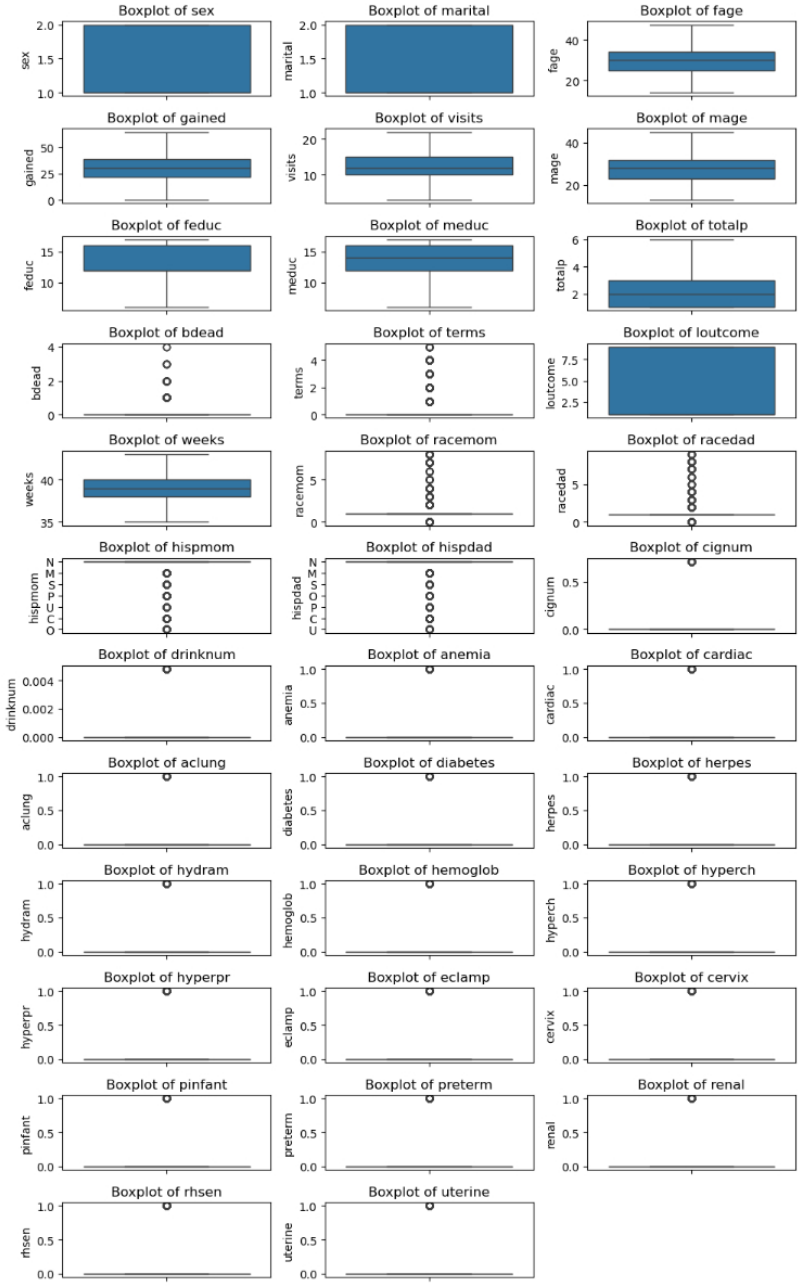
[52]:
```
# boxplot for all columns in the dataset to verify no outlier
plt.figure(figsize=(16, 8))
sns.boxplot(data=dataset.iloc[:, :-1]) # except last/object column
plt.title('Box Plots For all Features')
plt.xticks(rotation=45)
plt.ylabel('Values')
plt.xlabel('Columns')

# plt.savefig('single_boxplot_after_clip_outlier.png')
```

[52]:
```
Text(0.5, 0, 'Columns')
```

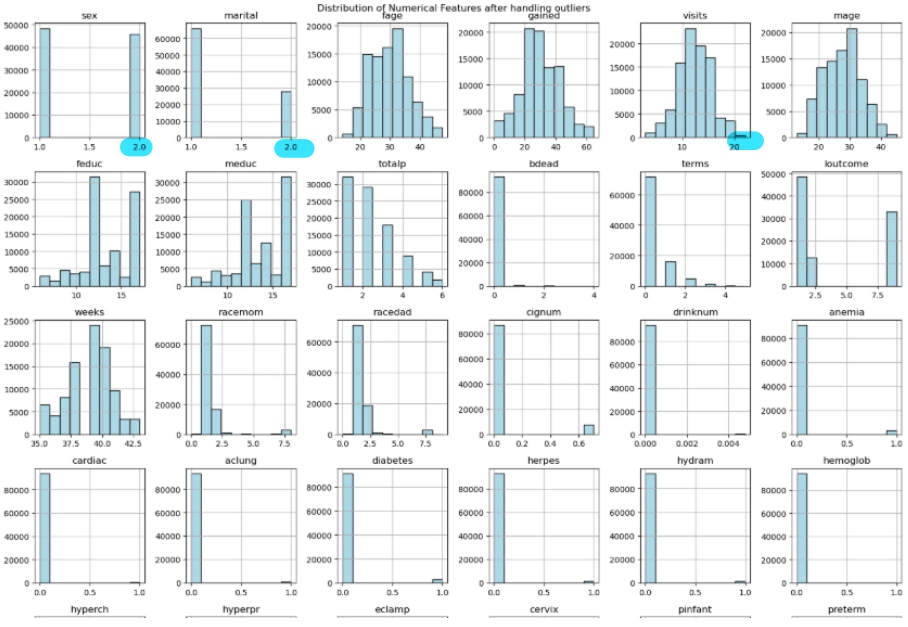


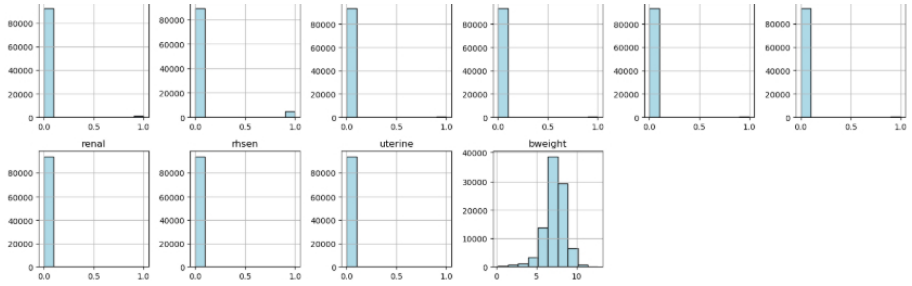
[53]:
```
plt.figure(figsize=(10, 16))
```

```
# boxplots for each column to verify no outlier
for i, column in enumerate(dataset.columns[:-1], 1):  # except the last/object column
    plt.subplot(12, 3, i)  # in 12 rows , 3 columns
    sns.boxplot(data=dataset, y=column)
    plt.title(f'Boxplot of {column}')
plt.tight_layout()  # spacing between subplots

# plt.savefig('individual_boxplots_after_outlier_handel.png')
```



[54]:
```
# Plot histograms for numerical features
numeric_columns = dataset.select_dtypes(include=['float64', 'int64']).columns
dataset[numeric_columns].hist(bins=10, figsize=(16, 16), color='lightblue', edgecolor='black')
plt.suptitle('Distribution of Numerical Features after handling outliers')
plt.tight_layout()
```



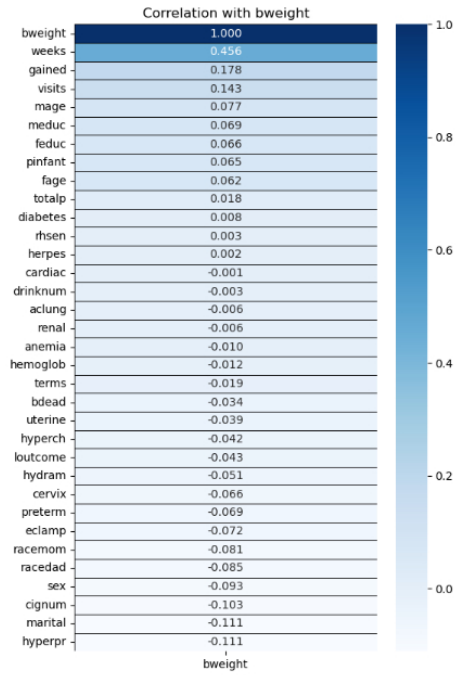


[55]:
```
# numeric columns
numeric_columns = dataset.select_dtypes(include=['number']).columns

# calc correlation
correlation_matrix = dataset[numeric_columns].corr()

# correlations with 'BWEIGHT'
correlation_with_bweight = correlation_matrix[['bweight']].sort_values(by='bweight', ascending=False)
correlation_with_bweight.head(15) # most 15 value has influence in bweight
```

[55]:

	bweight
bweight	1.000000
weeks	0.456367
gained	0.178322
visits	0.142505
mage	0.076865
meduc	0.068948
feduc	0.066069
pinfant	0.065061
fage	0.062490
totalp	0.017787
diabetes	0.007971
rhsen	0.002637
herpes	0.001570
cardiac	-0.000638
drinknum	-0.003348

[56]:
```
# heatmap
plt.figure(figsize=(6, 10))
sns.heatmap(correlation_with_bweight, annot=True, cmap='Blues', linewidths=0.5, linecolor='black', fmt='.3f')
plt.title('Correlation with bweight')
plt.yticks(rotation=0)

# Save the heatmap
plt.savefig('heatmap_bweight.png')
```



We can use this result to drop unnessessary column "not has influence in bweight" or for feature engineering "Create new features or transform existing ones"

Feature Engineering

Creating new features that enhance predictive power can capture more complex relationships in the data and improve model performance.

[59]:
```
# new columns can influence in bewieght
# average age of the parents might influence the bweight (older parents might have different health conditions or lifestyles)
dataset['avg_parent_age'] = (dataset['mage'] + dataset['fage']) / 2

# total education level of the parents might influence the bweight (more educated parents might follow healthcare)
dataset['total_parent_educ'] = dataset['meduc'] + dataset['feduc']

# provide insight about the mother health during pregnancy
dataset['gained_per_visit'] = dataset['gained'] / dataset['visits']

# to now category: very young or older mothers might have different risks
dataset['mage_category'] = pd.cut(dataset['mage'], bins=[0, 20, 30, 40, 50], labels=['<20', '20-30', '30-40', '>40'])

# more visits during a pregnancy - better health outcomes
dataset['visits_weeks_interaction'] = dataset['visits'] * dataset['weeks']

# combines multiple health-related columns into a single index that represents the overall health status of the mother
# number of health issues
dataset['mom_health_index'] = (
    dataset['anemia'] +
    dataset['cardiac'] +
    dataset['aclung'] +
    dataset['diabetes'] +
    dataset['herpes'] +
    dataset['hydram'] +
    dataset['hemoglob'] +
```

```
    dataset['hyperch'] +
    dataset['hyperpr'] +
    dataset['eclamp']
)

# Categorize the mother's health into groups based on the mom_health_index
dataset['mom_health_category'] = pd.cut(
    dataset['mom_health_index'],
    bins=[-1, 0, 1, 4, float('inf')],
    labels=['Good', 'Moderate', 'Poor', 'Very Poor']
)

# new features
dataset[['avg_parent_age','total_parent_educ','gained_per_visit','anemia','mage_category','visits_weeks_interaction', 'cardiac', 'aclung', 'diabetes', 'mom
```

[59]:

	avg_parent_age	total_parent_educ	gained_per_visit	anemia	mage_category	visits_weeks_interaction	cardiac	aclung	diabetes	mom_health_index	mom_health_categor
1	18.5	23.0	4.000000	0	<20	410.0	0	0	0	0.0	Goc
2	32.0	32.0	1.142857	0	30-40	546.0	0	0	0	0.0	Goc
3	26.5	24.0	2.666667	0	20-30	570.0	0	0	0	0.0	Goc
4	20.5	26.0	4.615385	0	<20	520.0	0	0	0	1.0	Moderat
5	21.0	25.0	2.000000	0	20-30	630.0	0	0	0	0.0	Goc

identify which features are most correlated with the bweight after make new features

[61]:
```
# numeric columns
numeric_columns = dataset.select_dtypes(include=['number']).columns

# calc correlation
correlation_matrix = dataset[numeric_columns].corr()
# correlation_matrix
```

[62]:
```
# correlations with bweight
correlation_with_bweight = correlation_matrix[['bweight']].sort_values(by='bweight', ascending=False)
correlation_with_bweight
```

[62]:

	bweight
bweight	1.000000
weeks	0.456367
visits_weeks_interaction	0.211882
gained	0.178322
visits	0.142505
mage	0.076865
avg_parent_age	0.073778
total_parent_educ	0.072631
meduc	0.068948
feduc	0.066069
pinfant	0.065061
fage	0.062490
gained_per_visit	0.053240
totalp	0.017787
diabetes	0.007971
rhsen	0.002637
herpes	0.001570
cardiac	-0.000638
drinknum	-0.003348
aclung	-0.005654
renal	-0.006348
anemia	-0.010373
hemoglob	-0.011676
terms	-0.019427
bdead	-0.033505
uterine	-0.038683
hyperch	-0.042369
loutcome	-0.042530
hydram	-0.050683
cervix	-0.065745
preterm	-0.068952
eclamp	-0.071985
racemom	-0.081421
racedad	-0.085485
mom_health_index	-0.092552
sex	-0.092687
cignum	-0.103420
marital	-0.110822
hyperpr	-0.111381

Columns with very low correlation (close to 0) or negative correlation with bweight can be removed Now I will make datset with fetures influence on bweight - according correlation , and logic knoweledge

[64]:
```
bweight_dataset = dataset[[
    'weeks',                # Strong correlation with bweight
    'visits_weeks_interaction', # Better health outcomes so influence on bweight
    'gained',               # Significant for bweight
    'visits',               # Indication of prenatal care
    'pinfant',              # Historical context, Mother had previous infant 4000+ grams
    'avg_parent_age',       # Lifestyle and care on mom and baby health
    'total_parent_educ',    # Total education level of parents
    'gained_per_visit',     # Weight gained per prenatal visit, insight into pregnancy health
    'mage_category',        # Mother's age category
    'mom_health_category', # Risk factors based on age groups
    'cignum',               # Risk factors
    'drinknum',             # Risk factors
    'diabetes',             # Presence of diabetes - dangerous on mom
    'mom_health_index',     # Health of mom issues
    'bweight'               # Target variable
]]

bweight_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 94054 entries, 1 to 101399
Data columns (total 15 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   weeks                    94054 non-null  float64
 1   visits_weeks_interaction  94054 non-null  float64
 2   gained                   94054 non-null  float64
 3   visits                   94054 non-null  int64
 4   pinfant                  94054 non-null  int64
 5   avg_parent_age           94054 non-null  float64
```

```
 6   total_parent_educ        94054 non-null  float64
 7   gained_per_visit         94054 non-null  float64
 8   mage_category            94054 non-null  category
 9   mom_health_category      94054 non-null  category
 10  cignum                   94054 non-null  float64
 11  drinknum                 94054 non-null  float64
 12  diabetes                 94054 non-null  int64
 13  mom_health_index         94054 non-null  float64
 14  bweight                  94054 non-null  float64
dtypes: category(2), float64(10), int64(3)
memory usage: 10.2 MB
```

[65]:
```
bweight_dataset.describe()
```

[65]:

	weeks	visits_weeks_interaction	gained	visits	pinfant	avg_parent_age	total_parent_educ	gained_per_visit	cignum	drinknum	9
count	94054.000000	94054.000000	94054.000000	94054.000000	94054.000000	94054.000000	94054.000000	94054.000000	94054.000000	94054.000000	
mean	38.908393	485.031142	30.026325	12.447317	0.005231	28.767054	26.581304	2.598156	0.056657	0.000011	
std	1.874216	130.968335	12.730399	3.259153	0.072137	5.799286	4.952360	1.435349	0.192167	0.000228	
min	35.000000	105.000000	0.000000	3.000000	0.000000	13.500000	12.000000	0.000000	0.000000	0.000000	
25%	38.000000	400.000000	22.000000	10.000000	0.000000	24.500000	24.000000	1.692308	0.000000	0.000000	
50%	39.000000	480.000000	30.000000	12.000000	0.000000	28.500000	26.000000	2.400000	0.000000	0.000000	
75%	40.000000	570.000000	39.000000	15.000000	0.000000	33.000000	31.000000	3.250000	0.000000	0.000000	
max	43.000000	946.000000	64.500000	22.000000	1.000000	46.250000	34.000000	21.000000	0.708433	0.004827	

[66]:
```
bweight_dataset['mom_health_category'].value_counts()
```

[66]:
```
mom_health_category
Good         79495
Moderate     12654
Poor          1905
Very Poor        0
Name: count, dtype: int64
```

Splitting the data into training and testing sets helps in evaluating the model's performance on unseen data, avoiding overfitting and ensuring generalization.

Split data to train and test

[68]:
```
from sklearn.model_selection import train_test_split

x = bweight_dataset.drop(columns=['bweight'])
y = bweight_dataset['bweight']

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
```

Data Transformation

Normalizing or scaling features is important to ensure that all features contribute equally to the model, especially when the features have different scales.

Standrization and normaliation for numeric values

[71]:
```
from sklearn.preprocessing import MinMaxScaler
# scaler
scaler = MinMaxScaler()
# numeric columns
numeric_columns = X_train.select_dtypes(include=['float64', 'int64']).columns

# Fit and transform the training data
X_train[numeric_columns] = scaler.fit_transform(X_train[numeric_columns])
# transform the testing data
X_test[numeric_columns] = scaler.transform(X_test[numeric_columns])

X_train.head(2)
```

[71]:

	weeks	visits_weeks_interaction	gained	visits	pinfant	avg_parent_age	total_parent_educ	gained_per_visit	mage_category	mom_health_category	cignum	di
93882	0.125	0.089180	0.201550	0.105263	0.0	0.229008	0.136364	0.123810	<20	Good	0.0	
13256	0.375	0.598098	0.465116	0.684211	0.0	0.687023	0.772727	0.089286	30-40	Good	0.0	

Encoding Categorical Variables

[73]:
```
from sklearn.preprocessing import OneHotEncoder

# OneHotEncoder
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
x_train_encoded = encoder.fit_transform(X_train[['mage_category', 'mom_health_category']])
x_test_encoded = encoder.transform(X_test[['mage_category', 'mom_health_category']])
```

[74]:
```
encoded_columns = encoder.get_feature_names_out(['mage_category', 'mom_health_category'])
encoded_columns
```

[74]:
```
array(['mage_category_20-30', 'mage_category_30-40', 'mage_category_<20',
       'mage_category_>40', 'mom_health_category_Good',
       'mom_health_category_Moderate', 'mom_health_category_Poor'],
      dtype=object)
```

[75]:
```
# x_train_encoded
```

[76]:
```
# addition -  retrive encoded data as df
X_train_encoded_df = pd.DataFrame(x_train_encoded, columns=encoded_columns, index=X_train.index)
X_test_encoded_df = pd.DataFrame(x_test_encoded, columns=encoded_columns, index=X_test.index)

# drop mage_category ,  mom_health_category  column and add encoded column each represent value
X_train = X_train.drop(['mage_category', 'mom_health_category'], axis=1).join(X_train_encoded_df)
X_test = X_test.drop(['mage_category', 'mom_health_category'], axis=1).join(X_test_encoded_df)
X_train.head(5)
```

[76]:

	weeks	visits_weeks_interaction	gained	visits	pinfant	avg_parent_age	total_parent_educ	gained_per_visit	cignum	drinknum	diabetes	mom_health_index
93882	0.125	0.089180	0.201550	0.105263	0.0	0.229008	0.136364	0.123810	0.0	0.0	0.0	0.0
13256	0.375	0.598098	0.465116	0.684211	0.0	0.687023	0.772727	0.089286	0.0	0.0	0.0	0.0
30989	0.625	0.588585	0.527132	0.631579	0.0	0.778626	0.954545	0.107937	0.0	0.0	0.0	0.0
33535	0.250	0.535077	0.759690	0.631579	0.0	0.106870	0.409091	0.155556	0.0	0.0	0.0	0.0
67958	0.500	0.338882	0.434109	0.368421	0.0	0.625954	1.000000	0.133333	0.0	0.0	0.0	0.0

Train Regressers

[78]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\hp\anaconda3\lib\site-packages (2.1.3)
Requirement already satisfied: numpy in c:\users\hp\anaconda3\lib\site-packages (from xgboost) (1.26.4)
Requirement already satisfied: scipy in c:\users\hp\anaconda3\lib\site-packages (from xgboost) (1.13.1)
```

[79]:
```
# dictionary to store RMSE values
rmse_values = {}
```

[80]:
```
import numpy as np
from sklearn.linear_model import Ridge, Lasso
from scipy.optimize import curve_fit
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
```

The Root Mean Squared Error (RMSE) is used to evaluate the accuracy of the models by measuring the difference between the actual and predicted values. It gives a clear indication of how well the model is performing.

• Linear Regression model "Lasso"

[82]:
```
# Lasso Regression
lasso_regression = Lasso(alpha=0.01)
lasso_regression.fit(X_train, y_train)
```

```
y_test_predict_lasso = lasso_regression.predict(X_test)
RMSE_lasso = np.sqrt(mean_squared_error(y_test, y_test_predict_lasso))
rmse_values['Linear Regression - Lasso'] = RMSE_lasso
print(f"Lasso Regression RMSE: {RMSE_lasso}")
```

```
Lasso Regression RMSE: 1.141701714501426 8
```

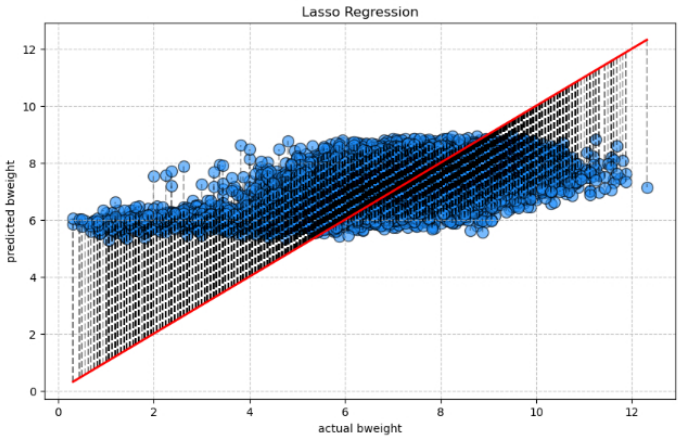
[83]:
```
results_df = pd.DataFrame({
    'Actual Value': y_test,
    'Predicted Value': y_test_predict_lasso
})

results_df.head(10)
```

[83]:

	Actual Value	Predicted Value
35304	9.1250	7.745116
77984	9.4375	6.370752
61265	8.2500	7.355441
49828	7.4375	7.189246
61851	8.3750	7.147922
6765	7.6250	7.097600
16206	6.0625	6.244758
64608	7.5000	8.189014
87712	6.9375	6.945761
33846	5.5000	6.658820

[84]:
```
plt.figure(figsize=(10, 6))

# actual values
plt.scatter(y_test, y_test_predict_lasso, color='dodgerblue', alpha=0.6, edgecolor='k', s=100, label='Predicted Values')

# link actual and predicted values with lines
for actual, predicted in zip(y_test, y_test_predict_lasso):
    plt.plot([actual, actual], [actual, predicted], 'k--', alpha=0.3)

# plot perfect fit line
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linewidth=2, label='Perfect Fit Line')

plt.grid(True, linestyle='--', alpha=0.7)
plt.xlabel('actual bweight')
plt.ylabel('predicted bweight')
plt.title('Lasso Regression')
```

[84]:
```
Text(0.5, 1.0, 'Lasso Regression')
```



• Linear Regression model "Ridge"

[86]:
```
# Ridge Regression
ridge_regression = Ridge()
ridge_regression.fit(X_train, y_train)
y_test_predict_ridge = ridge_regression.predict(X_test)
RMSE_ridge = np.sqrt(mean_squared_error(y_test, y_test_predict_ridge))
rmse_values['Linear Regression - Ridge'] = RMSE_ridge
print(f"Ridge Regression RMSE: {RMSE_ridge}")
```

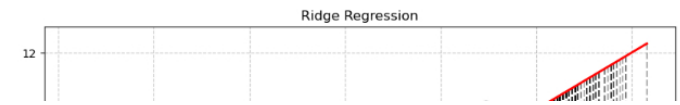
```
Ridge Regression RMSE: 1.1260093264996474
```

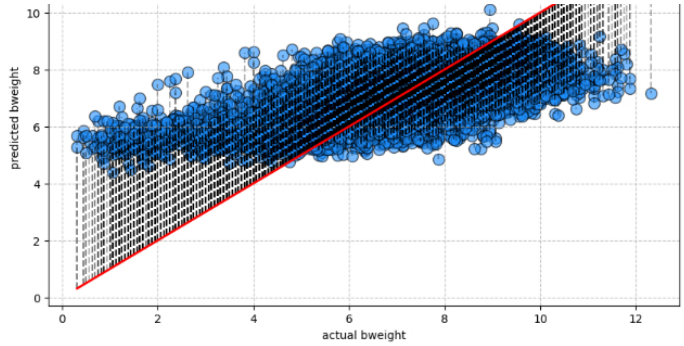
[87]:
```
results_df = pd.DataFrame({
    'Actual Value': y_test,
    'Predicted Value': y_test_predict_ridge
})

results_df.head(10)
```

[87]:

	Actual Value	Predicted Value
35304	9.1250	7.752874
77984	9.4375	6.743556
61265	8.2500	7.343751
49828	7.4375	7.155635
61851	8.3750	7.110418
6765	7.6250	6.925508
16206	6.0625	5.873396
64608	7.5000	8.263143
87712	6.9375	6.971915
33846	5.5000	6.805177

[88]:
```
plt.figure(figsize=(10, 6))

# actual values
plt.scatter(y_test, y_test_predict_ridge, color='dodgerblue', alpha=0.6, edgecolor='k', s=100, label='Predicted Values')

# link actual and predicted values with lines
for actual, predicted in zip(y_test, y_test_predict_ridge):
    plt.plot([actual, actual], [actual, predicted], 'k--', alpha=0.3)

# fit line
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linewidth=2, label='Perfect Fit Line')

plt.xlabel('actual bweight')
plt.ylabel('predicted bweight')
plt.title('Ridge Regression')
plt.grid(True, linestyle='--', alpha=0.7)
```





- fit_curve for linear regression

[90]:
```
# train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# equation
print("Linear Regression Equation:")
intercept = model.intercept_
coefficients = model.coef_
features = X_train.columns

equation_parts = [f"{coef} * {feature}" for coef, feature in zip(coefficients, features)]
equation = f"{intercept} + " + " + ".join(equation_parts)

equation
```

```
Linear Regression Equation:
```

[90]:
```
'2333847119327.875 + 3.8123461829624214 * weeks + -12.110715741227978 * visits_weeks_interaction + 0.9158277300170403 * gained + 11.06629692438277 * visits
+ 1.0224270506523205 * pinfant + 0.37322515205297946 * avg_parent_age + -0.060398331722995226 * total_parent_educ + 0.12695086308385395 * gained_per_visit
+ -0.3972266481529195 * cignum + -0.012927062105556211 * drinknum + 0.4750520895164958 * diabetes + -0.392888172400453 * mom_health_index + -241891473896.8
8135 * mage_category_20-30 + -241891473896.838 * mage_category_30-40 + -241891473897.04727 * mage_category_<20 + -241891473896.98535 * mage_category_>40 +
-2091955645426.1292 * mom_health_category_Good + -2091955645426.2793 * mom_health_category_Moderate + -2091955645426.4304 * mom_health_category_Poor'
```

[91]:
```
# predict values using test set
y_test_predict_linear = model.predict(X_test)

# RMSE
RMSE_linear = np.sqrt(mean_squared_error(y_test, y_test_predict_linear))
rmse_values['fit_curve - Linear Regression'] = RMSE_linear
print(f"Linear Regression RMSE: {RMSE_linear}")
```

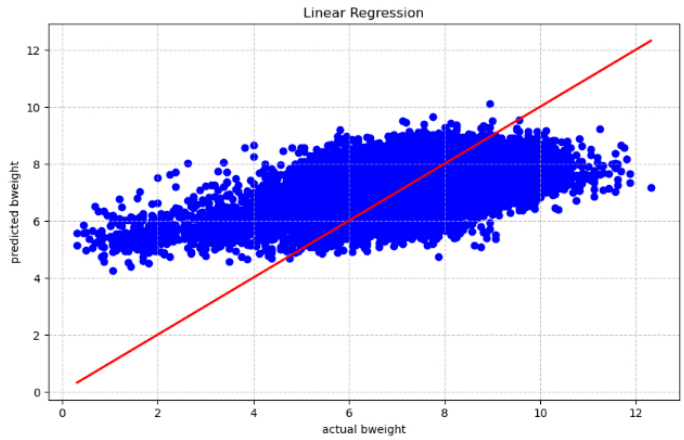
```
Linear Regression RMSE: 1.1251808101200584
```

[92]:
```
results_df = pd.DataFrame({
    'Actual Value': y_test,
    'Predicted Value': y_test_predict_linear
})

results_df.head(10)
```

[92]:

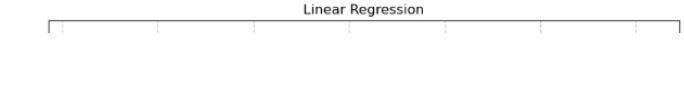
	Actual Value	Predicted Value
35304	9.1250	7.739746
77984	9.4375	6.742676
61265	8.2500	7.349121
49828	7.4375	7.150391
61851	8.3750	7.115723
6765	7.6250	6.929199
16206	6.0625	5.825195
64608	7.5000	8.266602
87712	6.9375	6.982422
33846	5.5000	6.802734

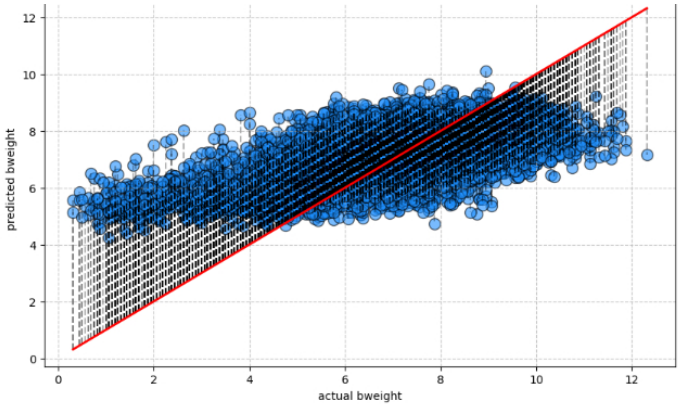
[93]:
```
# Scatter plot of actual vs predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_test_predict_linear, color='blue', label='Actual vs Predicted')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linewidth=2, label='Perfect Fit Line')
plt.xlabel('actual bweight')
plt.ylabel('predicted bweight')
plt.title('Linear Regression')
plt.grid(True, linestyle='--', alpha=0.7)
```



[94]:
```
plt.figure(figsize=(10, 6))

# scatterm actual values
plt.scatter(y_test, y_test_predict_linear, color='dodgerblue', alpha=0.6, edgecolor='k', s=100, label='Predicted Values')

# link actual and predicted values with lines
for actual, predicted in zip(y_test, y_test_predict_linear):
    plt.plot([actual, actual], [actual, predicted], 'k--', alpha=0.3)

# perfect fit line
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linewidth=2,  label='Perfect Fit Line')

plt.grid(True, linestyle='--', alpha=0.7)
plt.xlabel('actual bweight')
plt.ylabel('predicted bweight')
plt.title('Linear Regression')
```

[94]:
```
Text(0.5, 1.0, 'Linear Regression')
```





• fit_curve for non-linear regression

using Polynomial model

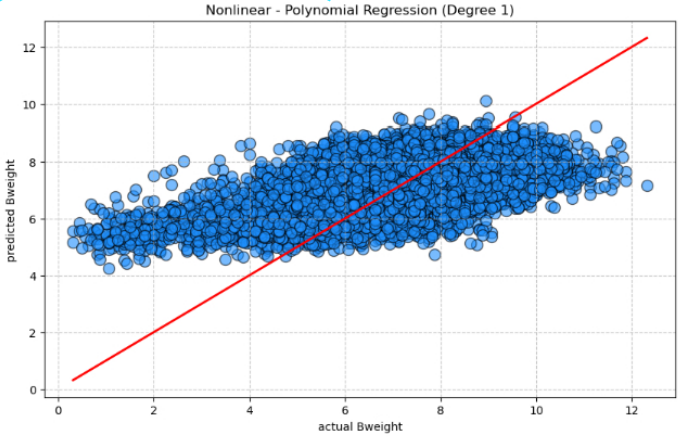
Polynomial model: $(y = a_0 + a_1x + \ldots + a_mx^m)$

[97]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

# polynomial regression for different degrees
def evaluate_polynomial_regression(X_train, X_test, y_train, y_test, max_degree):
    rmse_values = []
    for degree in range(1, max_degree + 1):
        # Polynomial regression model
        poly = PolynomialFeatures(degree=degree)
        poly_regression = make_pipeline(poly, LinearRegression())
        poly_regression.fit(X_train, y_train)

        # Predict values
        y_test_pred_poly = poly_regression.predict(X_test)

        # RMSE
        RMSE_poly = np.sqrt(mean_squared_error(y_test, y_test_pred_poly))
        rmse_values.append((degree, RMSE_poly))

        # Print results dataframe
        results_df = pd.DataFrame({
            'Actual Value': y_test,
            'Predicted Value': y_test_pred_poly
        })

        print(results_df.head(10))
        print(f"Polynomial Regression (Degree {degree}) RMSE: {RMSE_poly}")

        # Plot result
        plt.figure(figsize=(10, 6))
        plt.scatter(y_test, y_test_pred_poly, color='dodgerblue', alpha=0.6, edgecolor='k', s=100, label='Predicted Values')
        plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linewidth=2, label='Perfect Fit Line')
        plt.xlabel('actual Bweight')
        plt.ylabel('predicted Bweight')
        plt.title(f'Nonlinear - Polynomial Regression (Degree {degree})')
        plt.grid(True, linestyle='--', alpha=0.7)
        plt.show()

    return rmse_values
```

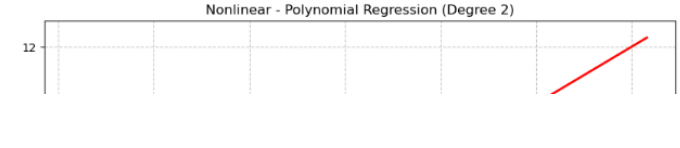
[98]:
```
# Polynomial regression for degrees from 1 to 3
max_degree = 3
rmse_values_poly = evaluate_polynomial_regression(X_train, X_test, y_train, y_test, max_degree)

# Find the degree with the minimum RMSE
best_degree, best_rmse = min(rmse_values_poly, key=lambda x: x[1])
rmse_values['fit_curve - NonLinear Regression - polynomial Model'] = best_rmse
print(f"The best degree for polynomial regression is: {best_degree} with RMSE: {best_rmse}")
```

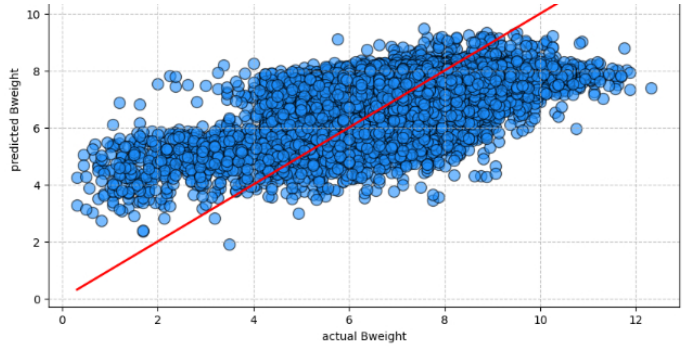
```
       Actual Value  Predicted Value
35304        9.1250         7.740479
77984        9.4375         6.742676
61265        8.2500         7.349365
49828        7.4375         7.150391
61851        8.3750         7.115723
6765         7.6250         6.930908
16206        6.0625         5.826660
64608        7.5000         8.266846
87712        6.9375         6.982910
33846        5.5000         6.802979
Polynomial Regression (Degree 1) RMSE: 1.1251847681587568
```

n = 1 same to linear



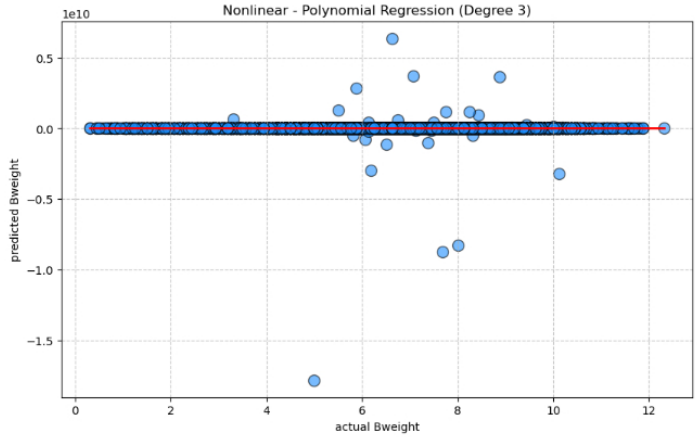

```
       Actual Value  Predicted Value
35304        9.1250         7.876110
77984        9.4375         6.322567
61265        8.2500         7.737377
49828        7.4375         7.383186
61851        8.3750         7.383308
6765         7.6250         7.250270
16206        6.0625         6.430278
64608        7.5000         7.748340
87712        6.9375         7.228153
33846        5.5000         6.712306
Polynomial Regression (Degree 2) RMSE: 1.0615413585607085
```





	Actual Value	Predicted Value
35304	9.1250	7.879822
77984	9.4375	6.513367
61265	8.2500	7.691467
49828	7.4375	7.337494
61851	8.3750	7.400879
6765	7.6250	7.328186
16206	6.0625	6.543152
64608	7.5000	7.596924
87712	6.9375	7.244293
33846	5.5000	6.545776

```
Polynomial Regression (Degree 3) RMSE: 141864725.78008565
```




```
The best degree for polynomial regression is: 2 with RMSE: 1.0615413585607085
```

using Power model

Power model: $(y = ax^b)$

[100]:
```
def power_func(x, a, b):
    return a * np.power(x, b)
```

[101]:
```
coefficients = []
equations = []
for feature in X_train.columns:
    # find the best parameters for the power function for each feature
    params, _ = curve_fit(power_func, X_train[feature], y_train)
    coefficients.append(params)
    a, b = params
    equation = f"y = {a:.4f} * {feature}^{b:.4f}"
    equations.append(equation)

# Print the equations for each feature
print("Power Regression Model Equations:")
for eq in equations:
    print(eq)
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_14296\2598982826.py:5: OptimizeWarning: Covariance of the parameters could not be estimated
  params, _ = curve_fit(power_func, X_train[feature], y_train)
Power Regression Model Equations:
y = 1.7410 * weeks^0.0000
y = 1.8979 * visits_weeks_interaction^0.0000
y = 7.2799 * gained^0.0000
y = 1.8016 * visits^0.0000
y = 8.4601 * pinfant^1.0000
y = 2.0187 * avg_parent_age^0.0000
y = 1.4723 * total_parent_educ^0.0000
y = 4.6525 * gained_per_visit^0.0000
y = 6.8242 * cignum^1.0000
y = 7.2605 * drinknum^1.0000
y = 7.3354 * diabetes^1.0000
y = 2.9642 * mom_health_index^0.0000
y = 7.2775 * mage_category_20-30^1.0000
y = 7.3678 * mage_category_30-40^1.0000
y = 7.0251 * mage_category_<20^1.0000
y = 7.1746 * mage_category_>40^1.0000
y = 7.3224 * mom_health_category_Good^1.0000
y = 7.0226 * mom_health_category_Moderate^1.0000
y = 6.7507 * mom_health_category_Poor^1.0000
```

[102]:
```
# Predict values
y_test_predict_power = np.zeros_like(y_test)
# predicted target values using the fitted power functions for each feature
for i, feature in enumerate(X_test.columns):
    a, b = coefficients[i]
    y_test_predict_power += power_func(X_test[feature], a, b) / len(X_test.columns)

# RMSE
RMSE_power = np.sqrt(mean_squared_error(y_test, y_test_predict_power))
rmse_values['fit_curve - Nonlinear Regression - Power Model'] = RMSE_power
print(f"Power Regression RMSE: {RMSE_power}")
```

```
Power Regression RMSE: 5.500985920382147
```

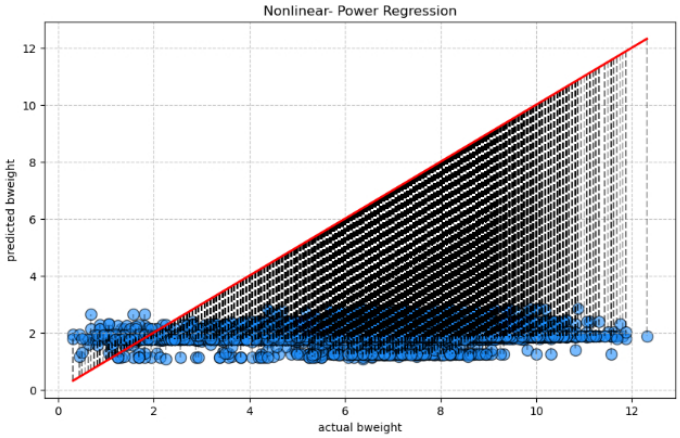
[103]:
```
results_df = pd.DataFrame({
    'Actual Value': y_test,
    'Predicted Value': y_test_predict_power
})

results_df.head(10)
```

[103]:

	Actual Value	Predicted Value
35304	9.1250	1.871269
77984	9.4375	2.397571
61265	8.2500	1.871269
49828	7.4375	1.866517
61851	8.3750	1.866517
6765	7.6250	2.011499
16206	6.0625	2.212402

64608	7.5000	1.871269
87712	6.9375	1.866517
33846	5.5000	1.866517

[104]:
```
plt.figure(figsize=(10, 6))

# Scatter plot of actual vs predicted values
plt.scatter(y_test, y_test_predict_power, color='dodgerblue', alpha=0.6, edgecolor='k', s=100, label='Predicted Values')

# Connect actual and predicted values with lines
for actual, predicted in zip(y_test, y_test_predict_power):
    plt.plot([actual, actual], [actual, predicted], 'k--', alpha=0.3)

# fit line
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linewidth=2,  label='Perfect Fit Line')

plt.xlabel('actual bweight')
plt.ylabel('predicted bweight')
plt.title('Nonlinear- Power Regression')
plt.grid(True, linestyle='--', alpha=0.7)
```



• Support Vector Machine (SVM) Regressor

Uses the rbf kernel to effectively model non-linear relationships while minimizing error within a margin of tolerance.

[106]:
```
# SVM Regressor
svm_regression = SVR(kernel='rbf')
# fit train data
svm_regression.fit(X_train, y_train)
# predict test data values
y_test_predict_svm = svm_regression.predict(X_test)
RMSE_svm = np.sqrt(mean_squared_error(y_test, y_test_predict_svm))
rmse_values['Nonlinear Regression - SVM'] = RMSE_svm
print(f"SVM Regressor RMSE: {RMSE_svm}")
```

```
SVM Regressor RMSE: 1.063992273771085
```

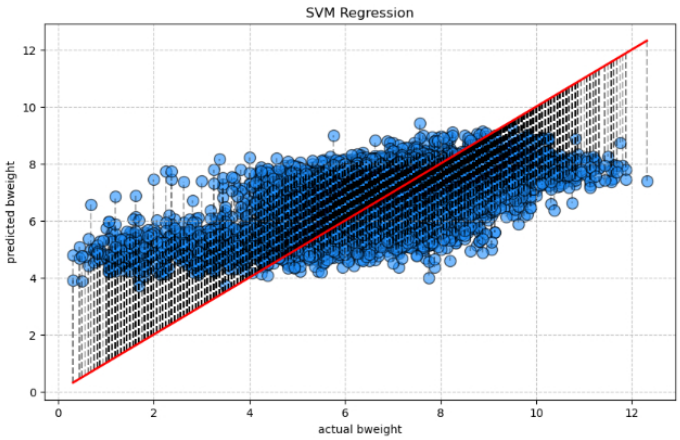
[107]:
```
results_df = pd.DataFrame({
    'Actual Value': y_test,
    'Predicted Value': y_test_predict_svm
})

results_df.head(10)
```

[107]:

	Actual Value	Predicted Value
35304	9.1250	7.866741
77984	9.4375	6.416038
61265	8.2500	7.705368
49828	7.4375	7.323704
61851	8.3750	7.372983
6765	7.6250	7.329343
16206	6.0625	6.206452
64608	7.5000	7.563322
87712	6.9375	7.207808
33846	5.5000	6.782067

[108]:
```
plt.figure(figsize=(10, 6))

# Scatter plot of actual vs predicted values
plt.scatter(y_test, y_test_predict_svm, color='dodgerblue', alpha=0.6, edgecolor='k', s=100, label='Predicted Values')

# Connect actual and predicted values with lines
for actual, predicted in zip(y_test, y_test_predict_svm):
    plt.plot([actual, actual], [actual, predicted], 'k--', alpha=0.3)

# perfect fit line
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linewidth=2, label='Perfect Fit Line')

plt.xlabel('actual bweight')
plt.ylabel('predicted bweight')
plt.grid(True, linestyle='--', alpha=0.7)
plt.title('SVM Regression')
```

[108]:
```
Text(0.5, 1.0, 'SVM Regression')
```



• XGBoost Regressor

Known for high performance and efficiency, it uses gradient boosting to optimize model performance through additive models.

[110]:
```
# XGBoost Regressor

# Convert feature names to strings and replace any special characters to make XGBRegressor can deal with
X_train.columns = [str(col).replace('[', '_').replace(']', '_').replace('<', '_') for col in X_train.columns]
X_test.columns = X_train.columns

xgboost_regression = XGBRegressor()
# fit train data values
xgboost_regression.fit(X_train, y_train)
# predict test data values
y_test_predict_xgboost = xgboost_regression.predict(X_test)
RMSE_xgboost = np.sqrt(mean_squared_error(y_test, y_test_predict_xgboost))
rmse_values['XGBoost Regresser'] = RMSE_xgboost
print(f"XGBoost Regressor RMSE: {RMSE_xgboost}")
```

```
XGBoost Regressor RMSE: 1.0597462440712795
```

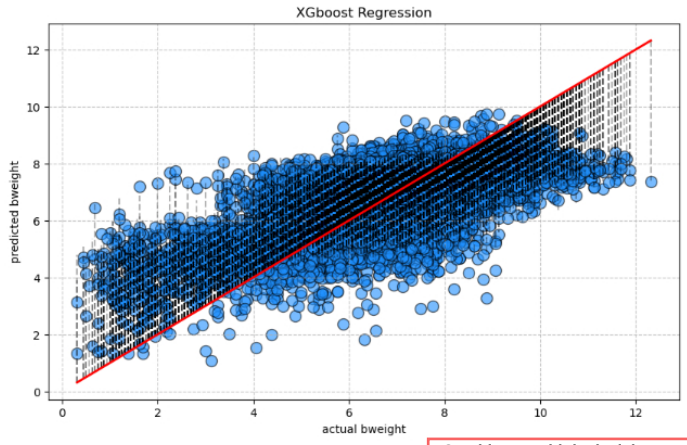
[111]:
```
results_df = pd.DataFrame({
    'Actual Value': y_test,
    'Predicted Value': y_test_predict_xgboost
})

results_df.head(10)
```

[111]:

	Actual Value	Predicted Value
35304	9.1250	7.817445
77984	9.4375	7.057749
61265	8.2500	7.686967
49828	7.4375	7.376152
61851	8.3750	7.305025
6765	7.6250	7.132318
16206	6.0625	6.246565
64608	7.5000	7.692577
87712	6.9375	7.212266
33846	5.5000	6.833161

[112]:
```
plt.figure(figsize=(10, 6))

# Scatter plot of actual vs predicted values
plt.scatter(y_test, y_test_predict_xgboost, color='dodgerblue', alpha=0.6, edgecolor='k', s=100, label='Predicted Values')

# Connect actual and predicted values with lines
for actual, predicted in zip(y_test, y_test_predict_svm):
    plt.plot([actual, actual], [actual, predicted], 'k--', alpha=0.3)

# perfect fit line
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linewidth=2, label='Perfect Fit Line')

plt.xlabel('actual bweight')
plt.ylabel('predicted bweight')
plt.grid(True, linestyle='--', alpha=0.7)
plt.title('XGboost Regression')
```

[112]:
```
Text(0.5, 1.0, 'XGboost Regression')
```



• Random Forest Regressor (with 50 estimators)

Combines multiple decision trees to handle non-linear relationships and interactions, providing robust predictions.

[114]:
```
rf_regression = RandomForestRegressor(n_estimators=50, random_state=42)
# fit train data values
rf_regression.fit(X_train, y_train)
# pridict test data values
y_test_predict_rf = rf_regression.predict(X_test)
RMSE_rf = np.sqrt(mean_squared_error(y_test, y_test_predict_rf))
rmse_values['Random Forest Regresser'] = RMSE_rf
print(f"Random Forest Regressor RMSE: {RMSE_rf}")
```

```
Random Forest Regressor RMSE: 1.0967006944764879
```

[115]:
```
results_df = pd.DataFrame({
    'Actual Value': y_test,
    'Predicted Value': y_test_predict_rf
})

results_df.head(10)
```

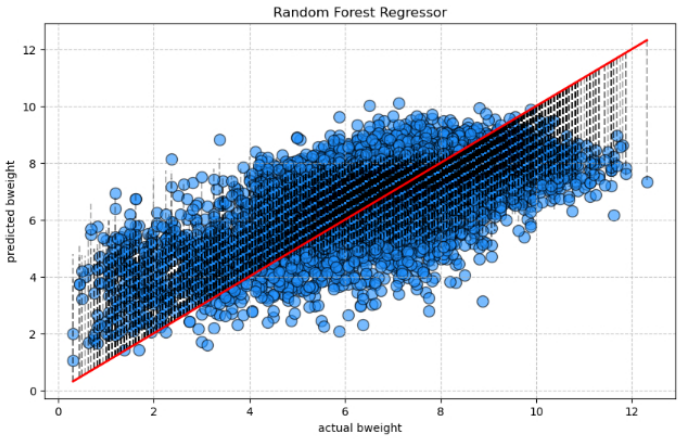
[115]:

	Actual Value	Predicted Value
35304	9.1250	8.027917
77984	9.4375	6.996562
61265	8.2500	8.102500
49828	7.4375	7.573750
61851	8.3750	6.692500
6765	7.6250	7.250000
16206	6.0625	5.597500
64608	7.5000	7.655000
87712	6.9375	7.265937
33846	5.5000	6.733125

[116]:
```
plt.figure(figsize=(10, 6))

# Scatter plot of actual vs predicted values
plt.scatter(y_test, y_test_predict_rf, color='dodgerblue', alpha=0.6, edgecolor='k', s=100, label='Predicted Values')

# Connect actual and predicted values with lines
for actual, predicted in zip(y_test, y_test_predict_svm):
    plt.plot([actual, actual], [actual, predicted], 'k--', alpha=0.3)

# perfect fit line
```

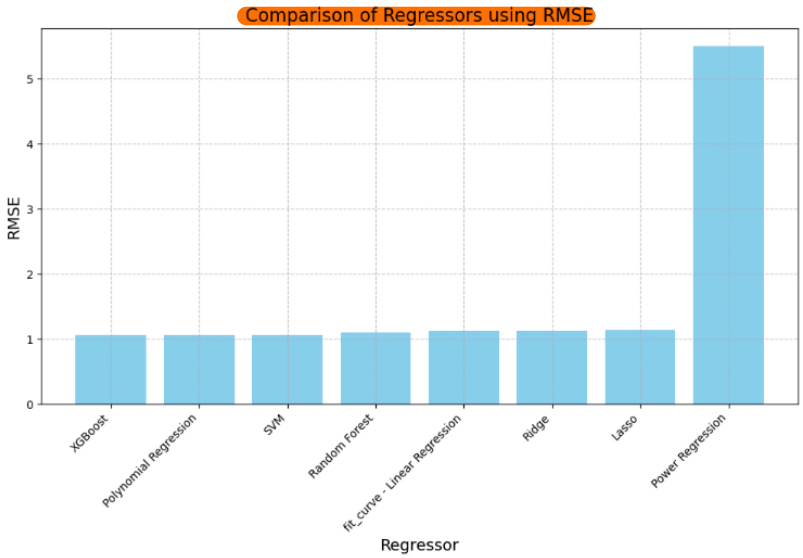
```
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linewidth=2, label='Perfect Fit Line')

plt.xlabel('actual bweight')
plt.ylabel('predicted bweight')
plt.grid(True, linestyle='--', alpha=0.7)
plt.title('Random Forest Regressor')
```

[116]:
```
Text(0.5, 1.0, 'Random Forest Regressor')
```



Evaluate Regressers

[118]:
```
# Create a mapping dictionary for new names
key_mapping = {
    'Linear Regression - Lasso': 'Lasso',
    'Linear Regression - Ridge': 'Ridge',
    'fit_curve - NonLinear Regression - polynomial Model': 'Polynomial Regression',
    'fit_curve - Nonlinear Regression - Power Model': 'Power Regression',
    'Nonlinear Regrission - SVM': 'SVM',
    'XGBoost Regresser': 'XGBoost',
    'Random Forest Regresser': 'Random Forest'
}

rmse_values_plot = {key_mapping.get(key, key): value for key, value in rmse_values.items()}

rmse_df = pd.DataFrame(list(rmse_values_plot.items()), columns=['Regressor', 'RMSE'])

rmse_df= rmse_df.sort_values(by='RMSE').reset_index(drop=True)

plt.figure(figsize=(12, 6))
plt.bar(rmse_df['Regressor'], rmse_df['RMSE'], color='skyblue')
plt.xlabel('Regressor', fontsize=14)
plt.ylabel('RMSE', fontsize=14)
plt.title('Comparison of Regressors using RMSE', fontsize=16)
plt.xticks(rotation=45, ha='right')
plt.grid(True, linestyle='--', alpha=0.7)
```



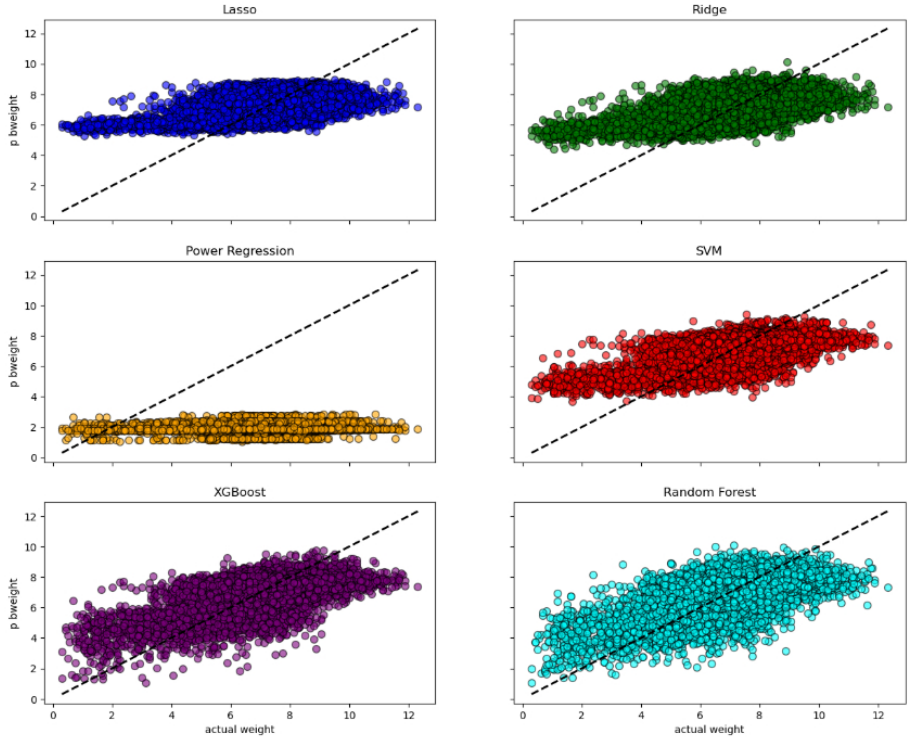
[119]:
```
rmse_df = pd.DataFrame(list(rmse_values.items()), columns=['Regressor', 'RMSE'])
rmse_df_sorted = rmse_df.sort_values(by='RMSE')
rmse_df_sorted
```

[119]:

	Regressor	RMSE
6	XGBoost Regresser	1.059746
3	fit_curve - NonLinear Regression - polynomial ...	1.061541
5	Nonlinear Regrission - SVM	1.063992
7	Random Forest Regresser	1.096701
2	fit_curve - Linear Regression	1.125181
1	Linear Regression - Ridge	1.126009
0	Linear Regression - Lasso	1.141702
4	fit_curve - Nonlinear Regression - Power Model	5.500986

[120]:
```
fig, axs = plt.subplots(3, 2, figsize=(15, 12), sharex=True, sharey=True)

# Linear Regression (Lasso)
axs[0, 0].scatter(y_test, y_test_predict_lasso, color='blue', alpha=0.6, edgecolor='k', s=50, label='Lasso')
axs[0, 0].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='black', linewidth=2, linestyle='--')
axs[0, 0].set_title('Lasso')

# Linear Regression (Ridge)
axs[0, 1].scatter(y_test, y_test_predict_ridge, color='green', alpha=0.6, edgecolor='k', s=50, label='Ridge')
axs[0, 1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='black', linewidth=2, linestyle='--')
axs[0, 1].set_title('Ridge')

# NonLinear - Power Regression
axs[1, 0].scatter(y_test, y_test_predict_power, color='orange', alpha=0.6, edgecolor='k', s=50, label='Power Regression')
axs[1, 0].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='black', linewidth=2, linestyle='--')
axs[1, 0].set_title('Power Regression')

# NonLinear SVM Regressor
axs[1, 1].scatter(y_test, y_test_predict_svm, color='red', alpha=0.6, edgecolor='k', s=50, label='SVM')
```

```
axs[1, 1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='black', linewidth=2, linestyle='--')
axs[1, 1].set_title('SVM')

# XGBoost Regressor
axs[2, 0].scatter(y_test, y_test_predict_xgboost, color='purple', alpha=0.6, edgecolor='k', s=50, label='XGBoost')
axs[2, 0].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='black', linewidth=2, linestyle='--')
axs[2, 0].set_title('XGBoost')

# Random Forest Regressor
axs[2, 1].scatter(y_test, y_test_predict_rf, color='cyan', alpha=0.6, edgecolor='k', s=50, label='Random Forest')
axs[2, 1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='black', linewidth=2, linestyle='--')
axs[2, 1].set_title('Random Forest')

# common Labels
for ax in axs.flat:
    ax.set(xlabel='actual weight', ylabel='p bweight')
    ax.label_outer()

fig.suptitle('Comparison of Different Regressors', fontsize=16)

# Show Legend
handles, labels = axs[0, 0].get_legend_handles_labels()
plt.savefig('all_Regressors.png')
```



[ ]: