Finding the strenght of Cement

Importing Libraries

In [1]:
```
from pycaret.regression import *
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import scipy.stats as stats
from sklearn.preprocessing import StandardScaler
import joblib
```

In [2]:
```
df=pd.read_csv('concrete.csv')
df.head()
```

Out[2]:

	cement	slag	ash	water	superplastic	coarseagg	fineagg	age	strength
0	141.3	212.0	0.0	203.5	0.0	971.8	748.5	28	29.89
1	168.9	42.2	124.3	158.3	10.8	1080.8	796.2	14	23.51
2	250.0	0.0	95.7	187.4	5.5	956.9	861.2	28	29.22
3	266.0	114.0	0.0	228.0	0.0	932.0	670.0	28	45.85
4	154.8	183.4	0.0	193.3	9.1	1047.4	696.7	28	18.29

EDA

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1030 entries, 0 to 1029
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   cement        1030 non-null   float64
 1   slag          1030 non-null   float64
 2   ash           1030 non-null   float64
 3   water         1030 non-null   float64
 4   superplastic  1030 non-null   float64
 5   coarseagg     1030 non-null   float64
 6   fineagg       1030 non-null   float64
 7   age           1030 non-null   int64
 8   strength      1030 non-null   float64
dtypes: float64(8), int64(1)
memory usage: 72.5 KB
```

In [4]:
```
df.describe()
```

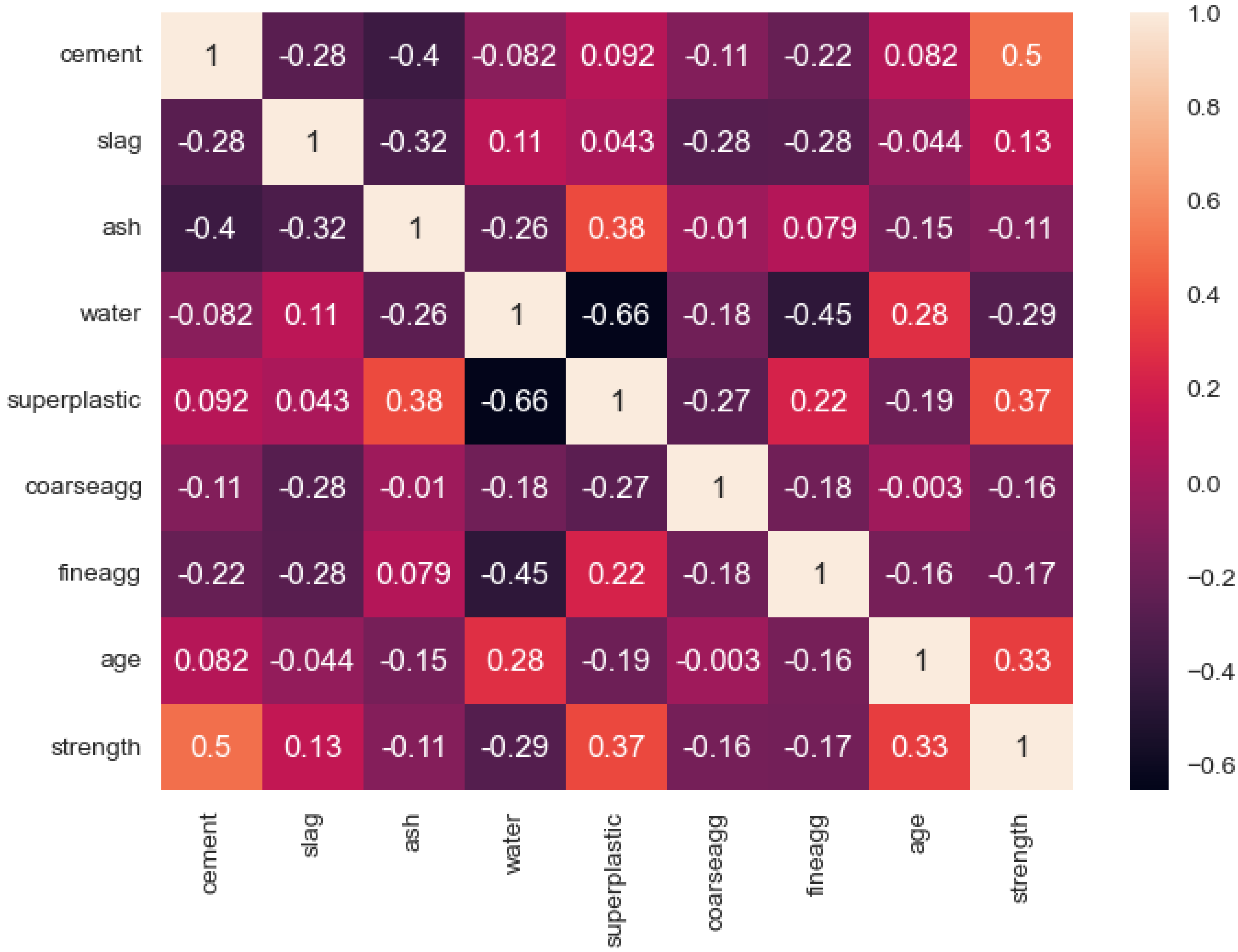
Out[4]:

	cement	slag	ash	water	superplastic	coarseagg	fineagg	age
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000
mean	281.167864	73.895825	54.188350	181.567282	6.204660	972.918932	773.580485	45.662136
std	104.506364	86.279342	63.997004	21.354219	5.973841	77.753954	80.175980	63.169912

min	102.000000	0.000000	0.000000	121.800000	0.000000	801.000000	594.000000	1.000000
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.950000	7.000000
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968.000000	779.500000	28.000000
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.400000	824.000000	56.000000
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.600000	365.000000

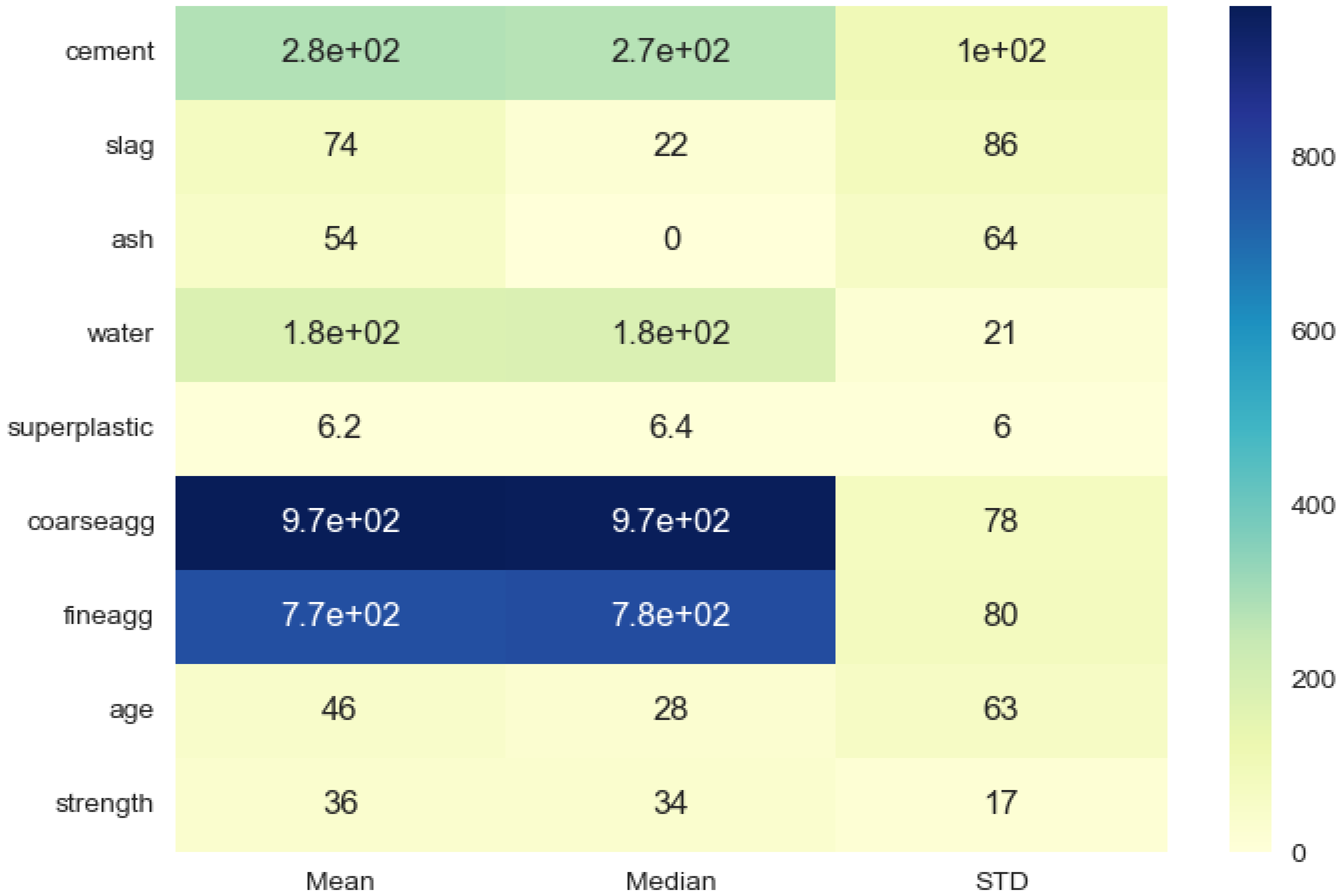
In [5]:

```
sns.heatmap(df.corr(),annot=True,cmap="rocket")
plt.show()
```



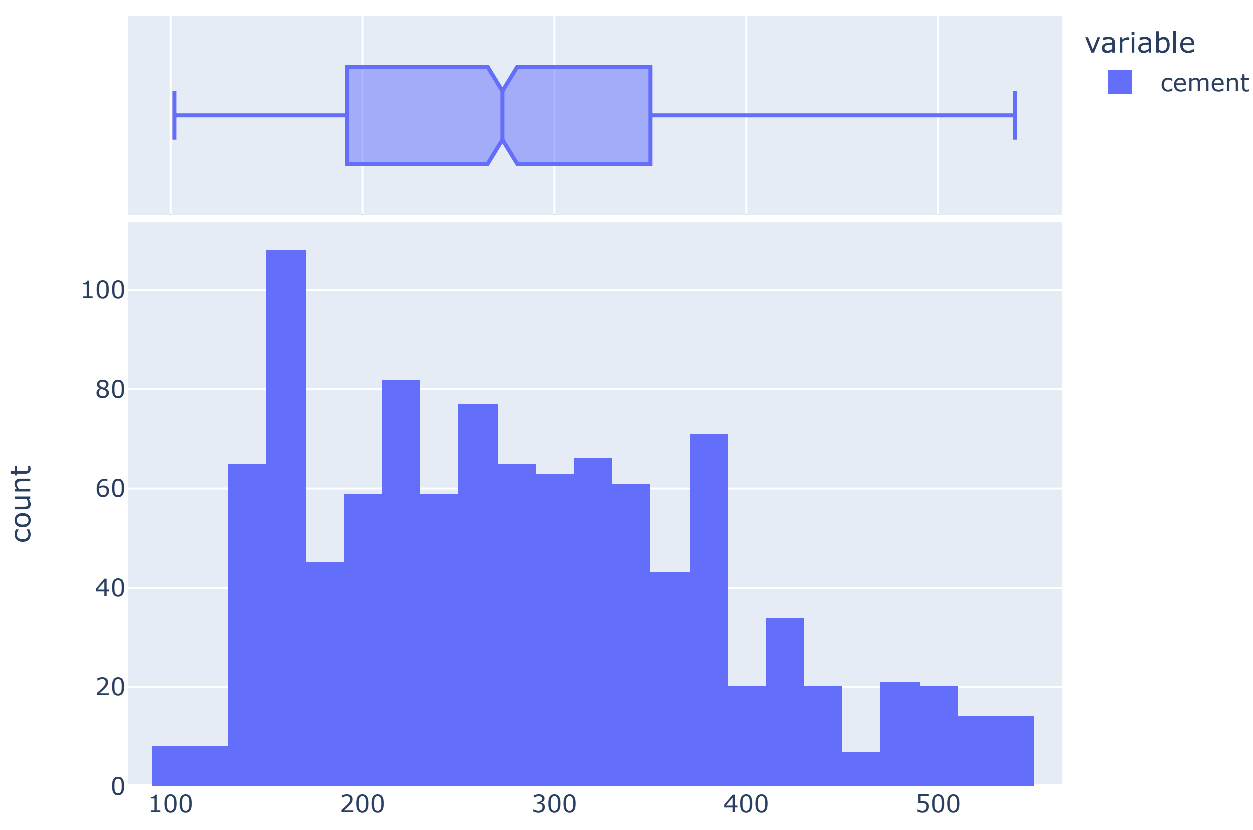
In [6]:

```
mean=df.mean()
median=df.median()
std=df.std()
a=pd.DataFrame({'Mean':mean.tolist(),'Median':median.tolist(),'STD':std.tolist()},index=
sns.heatmap(a,annot=True,cmap='YlGnBu')
plt.show()
```



Cement

In [7]:
```
fig=px.histogram(df.cement,marginal='box')
fig.show()
```



value

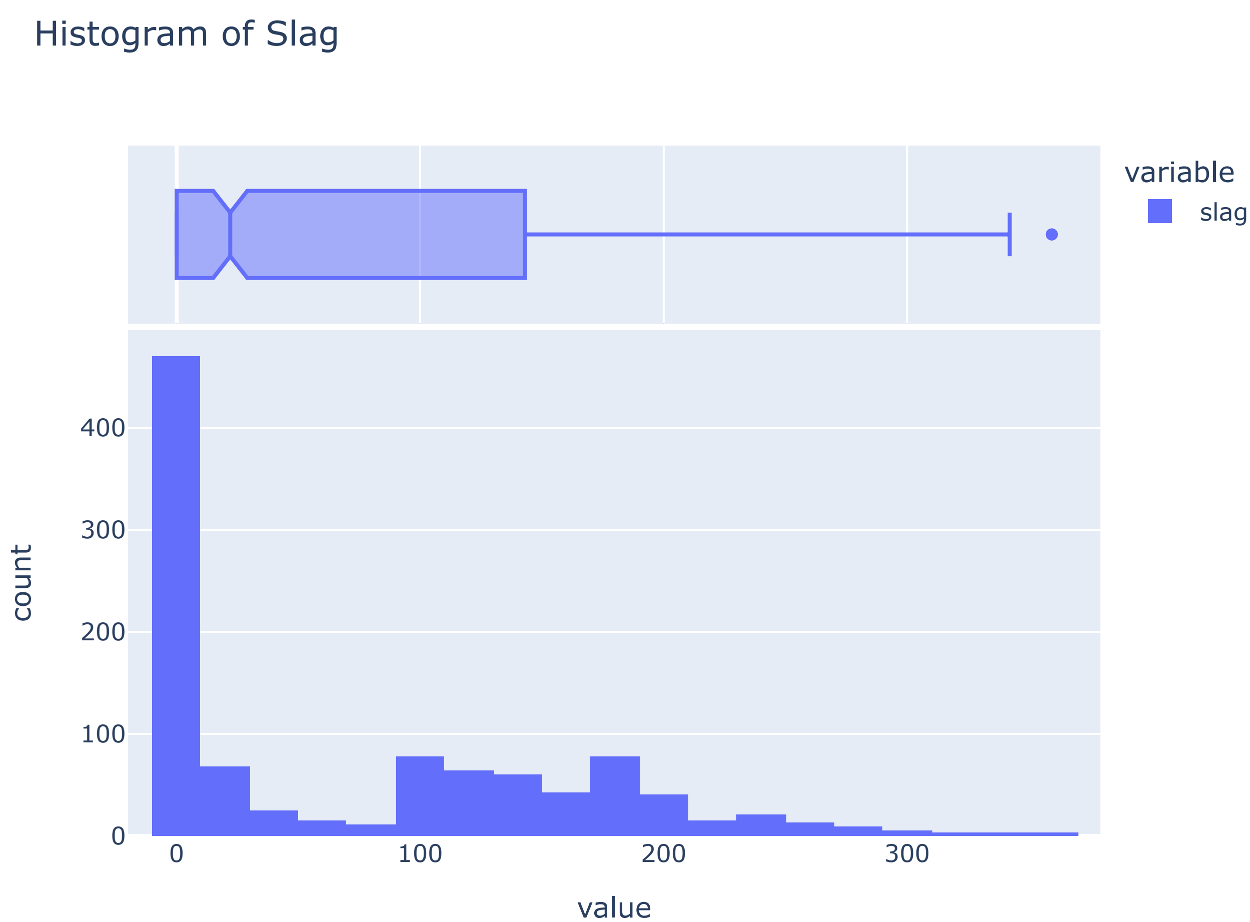
In [8]:
```python
def detect_outlier_z_score(data,threshold=3):
    mean=np.mean(data)
    std=np.std(data)
    z_scores=[(x-mean)/std for x in data]
    outliers=[x for i,x in enumerate(data) if np.abs(z_scores[i])>threshold]
    return outliers

outliers=detect_outlier_z_score(df.cement,3)
print(outliers)
```

```
[]
```

No outliers

Slag

In [9]:
```python
fig=px.histogram(df.slag,title="Histogram of Slag",marginal="box")
fig.show()
```



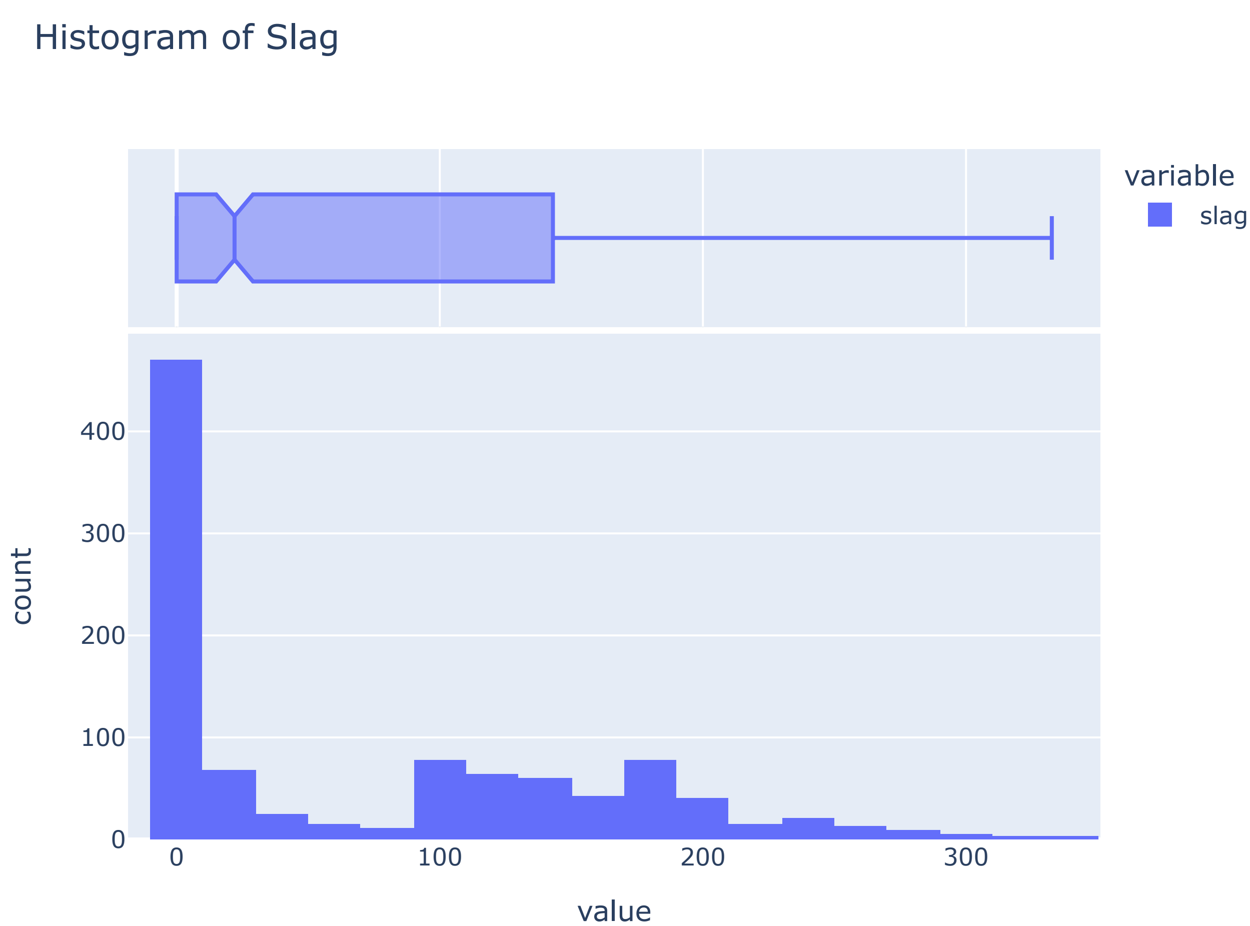
In [10]:
```python
def detect_outlier_z_score(data,threshold=3):
    mean=np.mean(data)
    std=np.std(data)
    z_scores=[(x-mean)/std for x in data]
    outliers=[]
    for i,x in enumerate(data):
        if z_scores[i]>threshold or z_scores[i]<-threshold:
```

```
            outliers.append(x)
    return outliers
outliers=detect_outlier_z_score(df.slag,3)
print(outliers)
```

```
[342.1, 342.1, 359.4, 359.4]
```

In [11]:
```
def fix_z_score(data,threshold=3):
    mean=np.mean(data)
    std=np.std(data)
    upper_bound=mean+threshold*std
    lower_bound=mean-threshold*std
    z_scores=[(x-mean)/std for x in data]
    for i,x in enumerate(data):
        if z_scores[i]>threshold:
            data[i]=upper_bound
        if z_scores[i]<-threshold:
            data[i]=lower_bound
fix_z_score(df.slag,3)
```

In [12]:
```
fig=px.histogram(df.slag,title="Histogram of Slag",marginal="box")
fig.show()
```



In [13]:
```
outliers=fix_z_score(df.ash,3)
print(outliers)
```

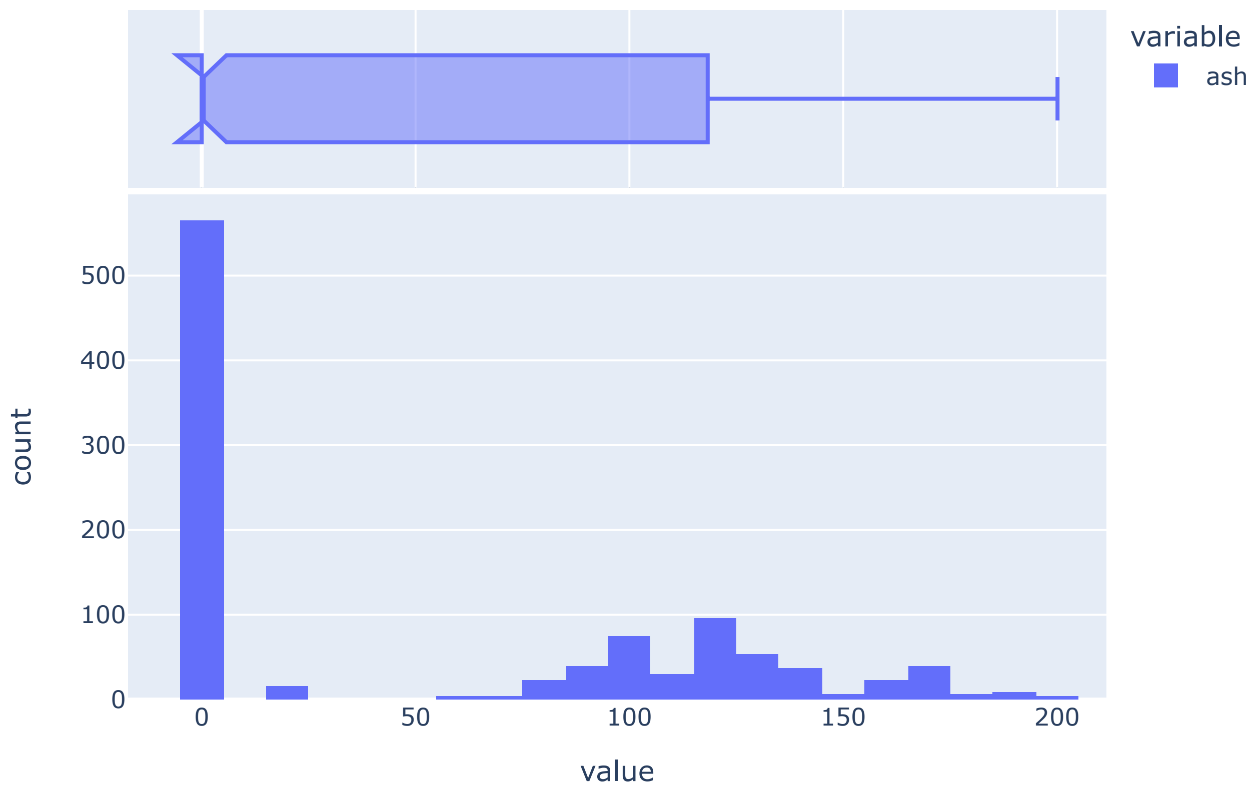
```
None
```

Ash

In [14]:
```
fig=px.histogram(df.ash,title="Histogram of Ash",marginal="box")
```

```
fig.show()
```

Histogram of Ash



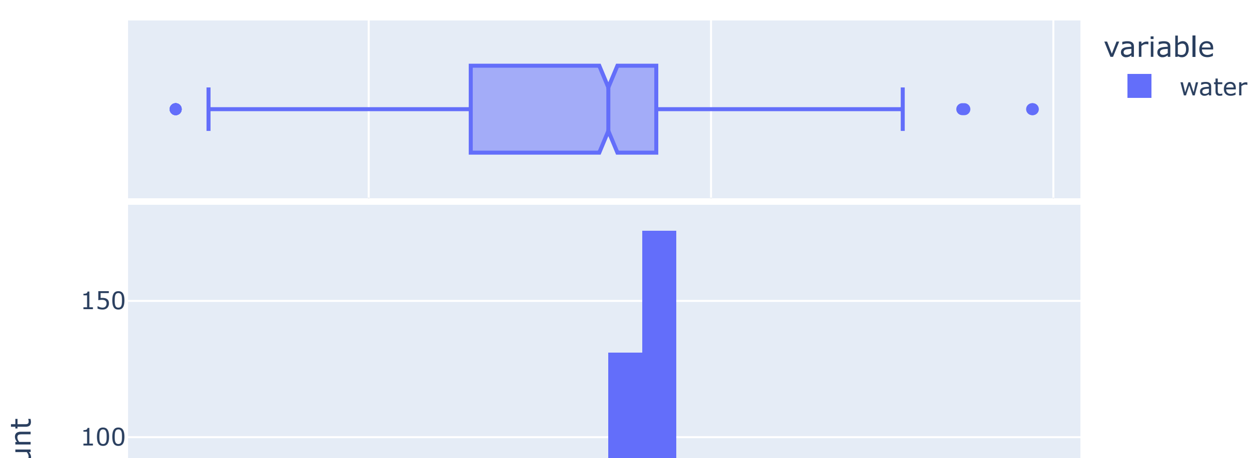
In [15]:
```
outliers=detect_outlier_z_score(df.ash,3)
print(outliers)
```

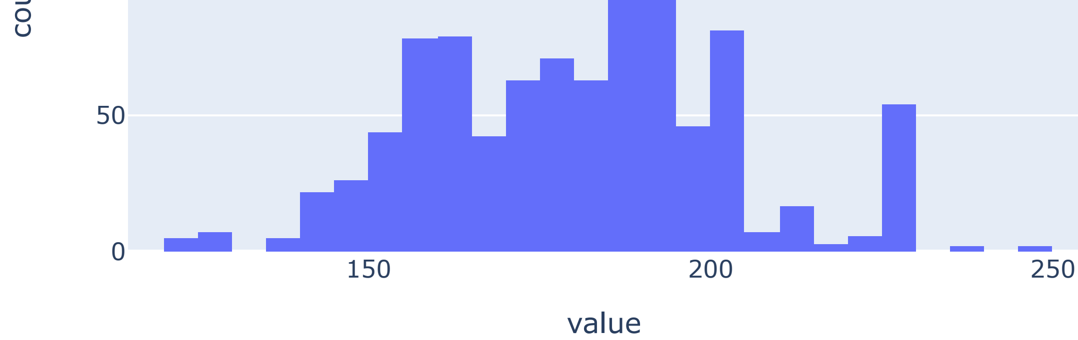
```
[]
```

Water

In [16]:
```
fig=px.histogram(df.water,title="Histogram of Water",marginal="box")
fig.show()
```

Histogram of Water





In [17]:
```
fix_z_score(df.water,3)
```

In [18]:
```
print(np.mean(df.water)+3*np.std(df.water))
```

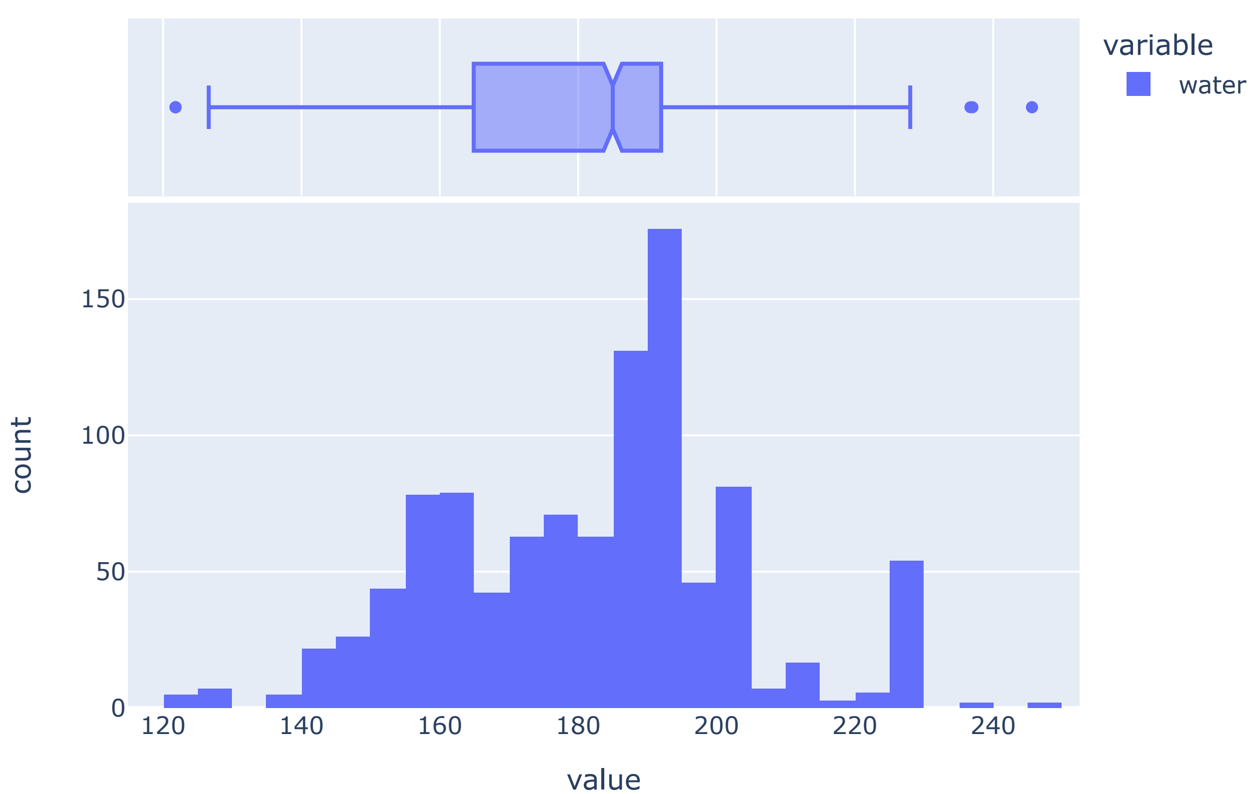
```
245.57234062956732
```

In [19]:
```
outliers=detect_outlier_z_score(df.water,3)
print(outliers)
```

```
[245.59883132012828, 245.59883132012828]
```

In [20]:
```
fig=px.histogram(df.water,title="Histogram of Water",marginal="box")
fig.show()
```

Histogram of Water

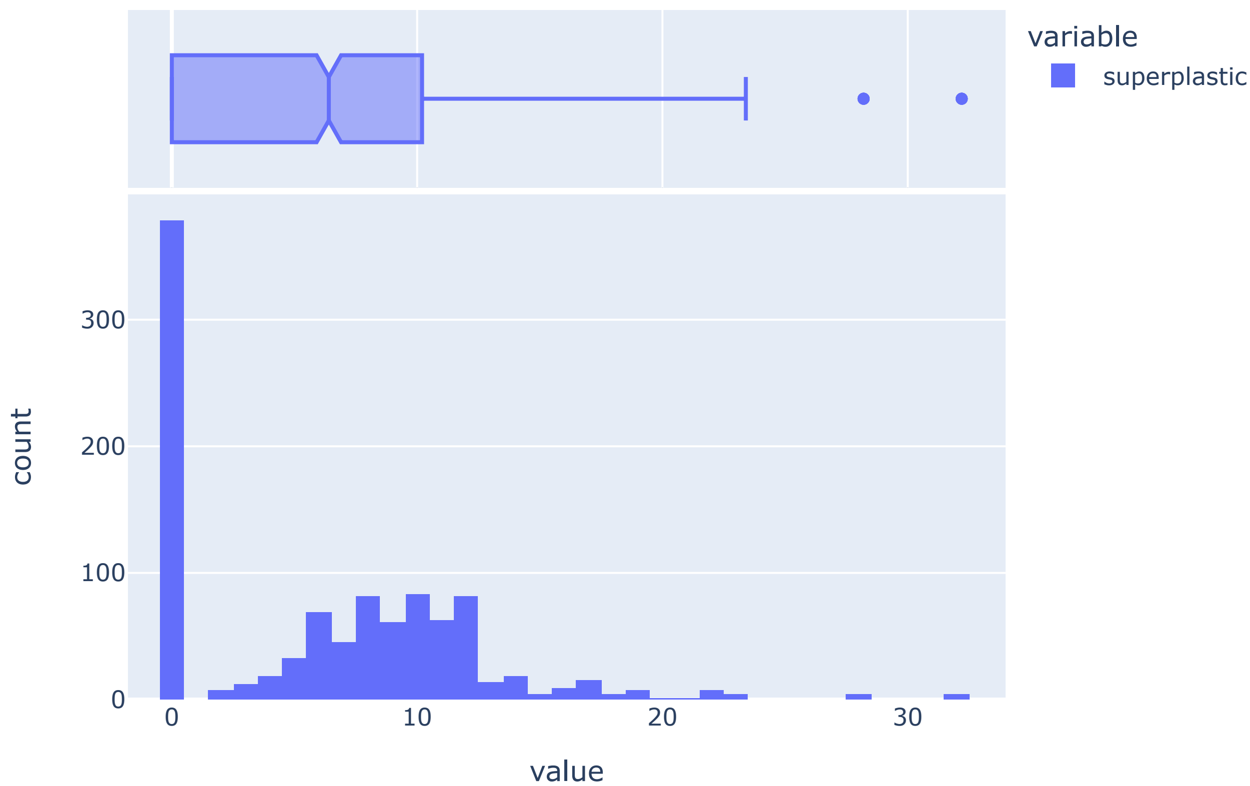


Superplastic

In [21]:
```
fig=px.histogram(df.superplastic,title="Histogram of Superplastic",marginal="box")
```

```python
fig.show()
```

Histogram of Superplastic

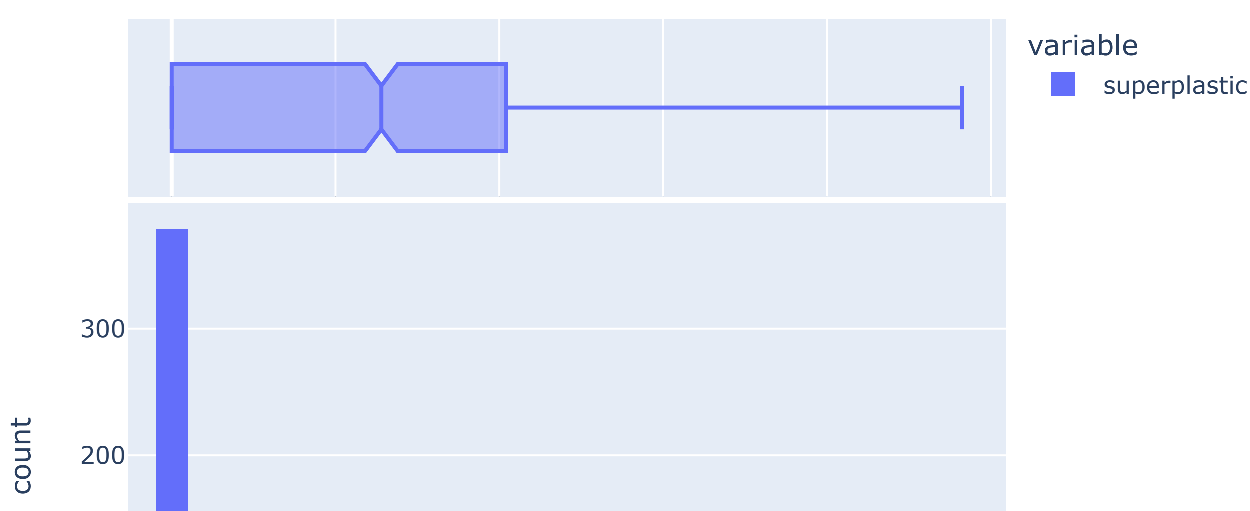


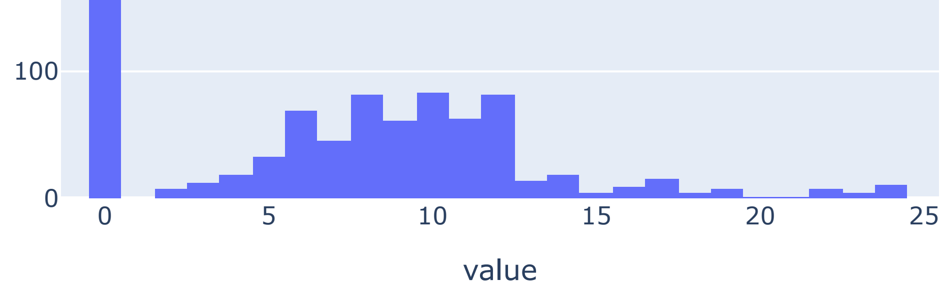

```python
In [22]: outliers=detect_outlier_z_score(df.superplastic,3)
print(outliers)
```

```
[28.2, 28.2, 32.2, 32.2, 28.2, 32.2, 32.2, 28.2, 32.2, 28.2]
```

```python
In [23]: fix_z_score(df.superplastic,3)
fig=px.histogram(df.superplastic,title="Histogram of Superplastic",marginal="box")
fig.show()
```

Histogram of Superplastic

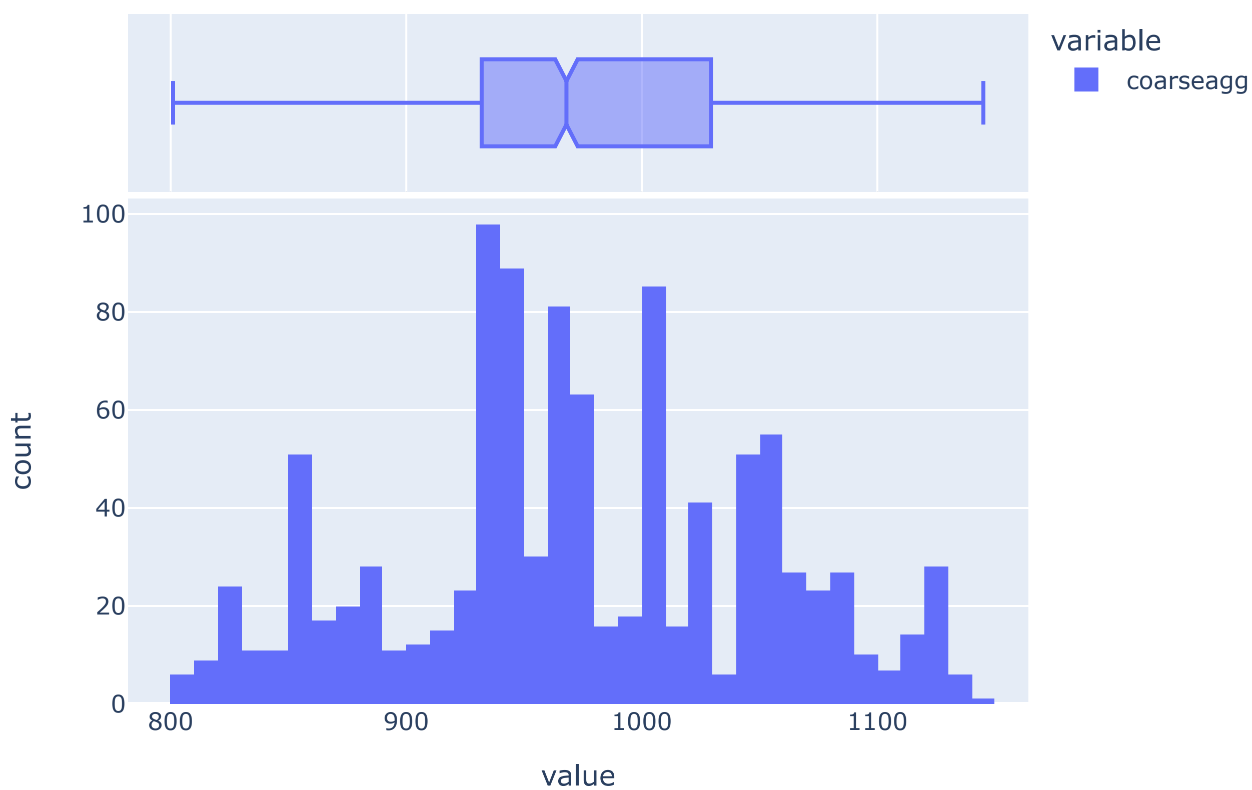




Coarseagg

In [24]:
```
fig=px.histogram(df.coarseagg,title="Histogram of Coarseagg",marginal="box")
fig.show()
```

Histogram of Coarseagg



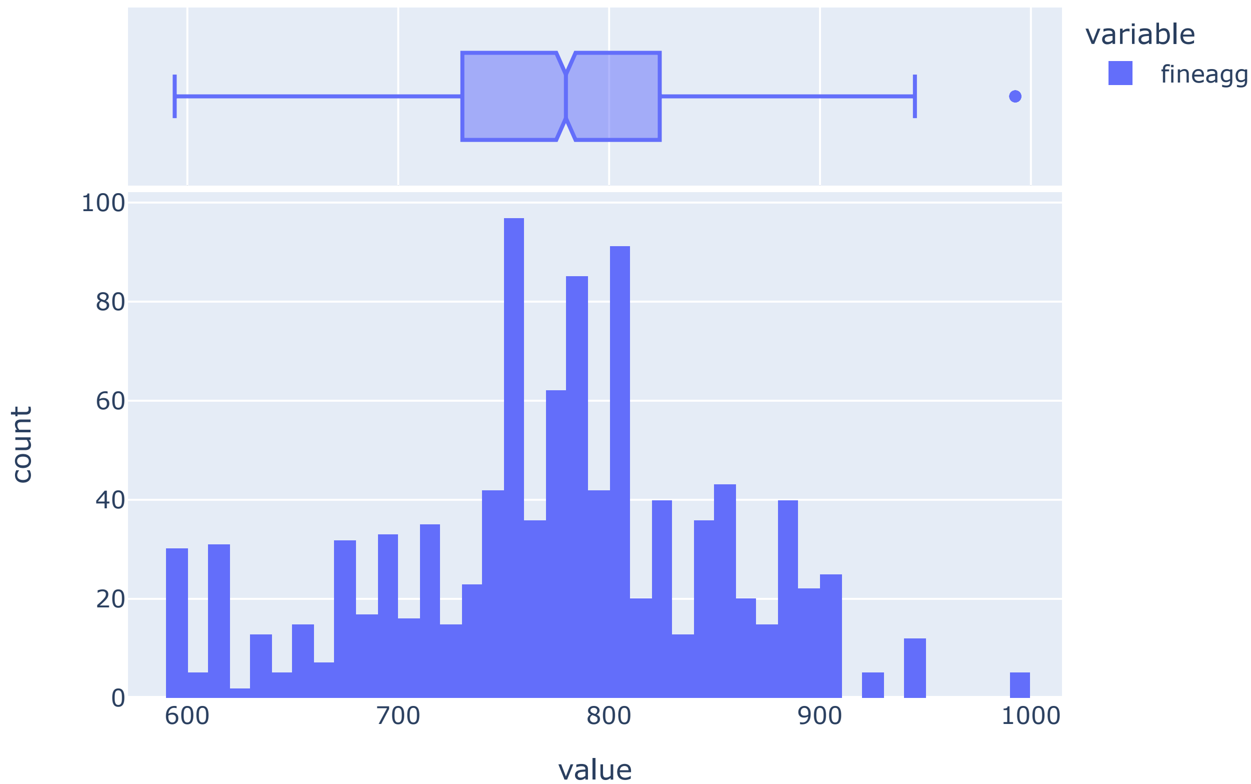
In [25]:
```
outliers=detect_outlier_z_score(df.ash,3)
print(outliers)
```

```
[]
```

Fineagg

In [26]:
```
fig=px.histogram(df.fineagg,title="Histogram of Fineagg",marginal="box")
fig.show()
```

Histogram of Fineagg



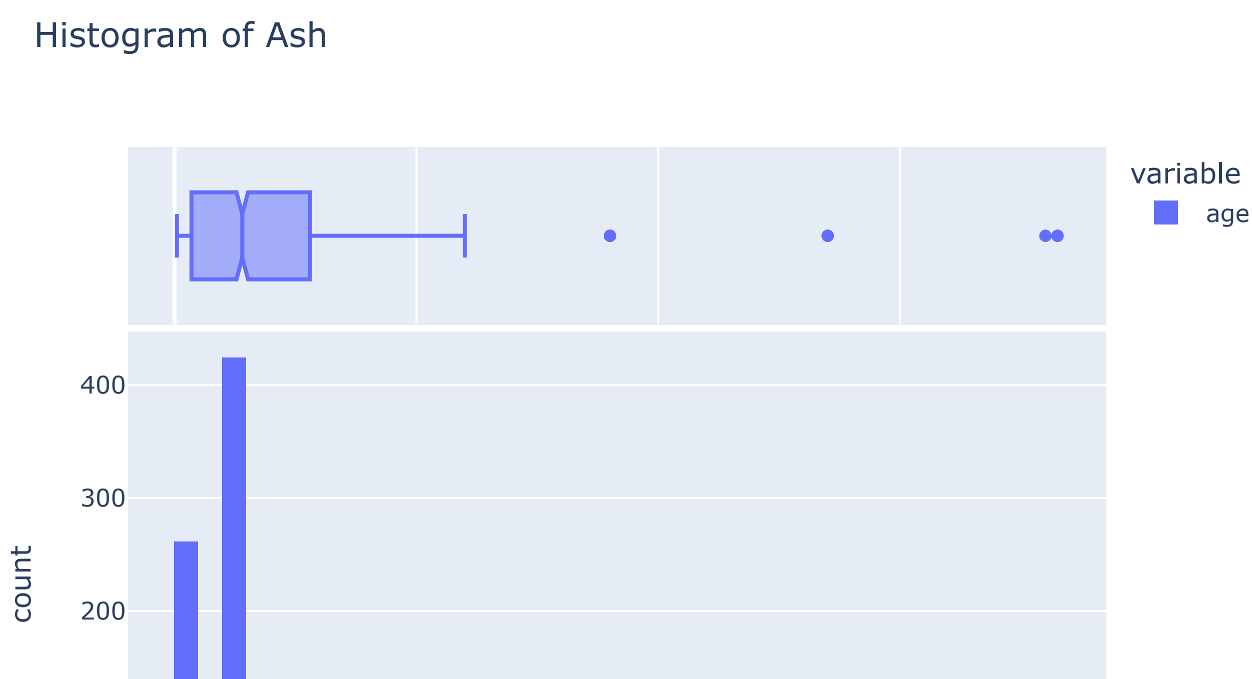
In [27]:
```
outliers=detect_outlier_z_score(df.fineagg,3)
print(outliers)
```

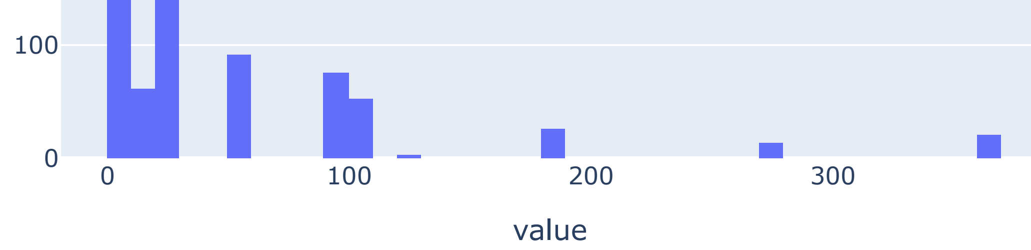
```
[]
```

In [ ]:

age

In [28]:
```
fig=px.histogram(df.age,title="Histogram of Ash",marginal="box")
fig.show()
```



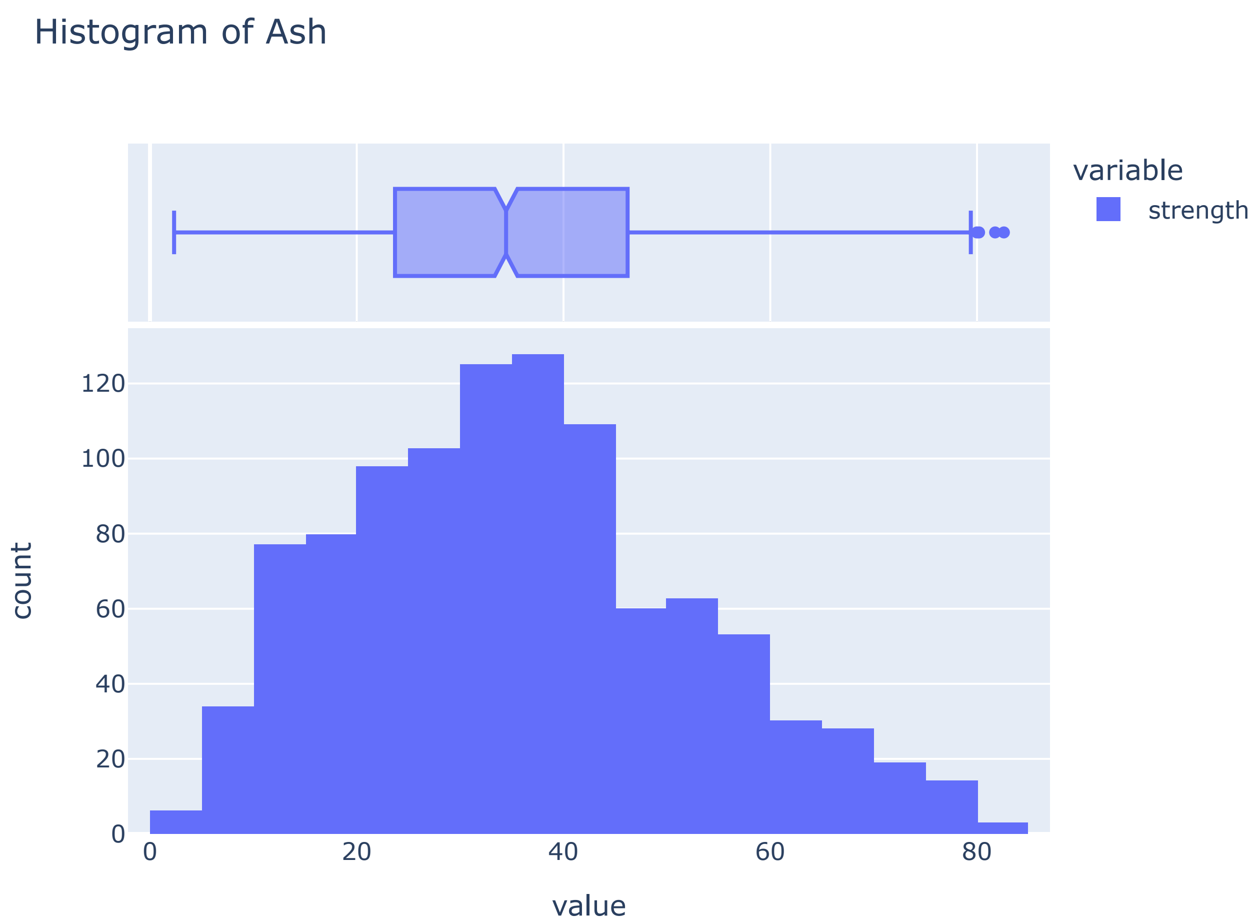


In [29]:
```
outliers=detect_outlier_z_score(df.age,3)
print(outliers)
```

```
[365, 365, 270, 360, 365, 365, 270, 270, 270, 270, 270, 360, 360, 365, 360, 365, 365, 27
0, 365, 270, 270, 365, 365, 365, 360, 270, 270, 365, 360, 270, 365, 365, 270]
```

Strenght

In [30]:
```
fig=px.histogram(df.strength,title="Histogram of Ash",marginal="box")
fig.show()
```



In [31]:
```
outliers=detect_outlier_z_score(df.strength,3)
print(outliers)
```

```
[]
```

In [215…
```
df.describe()
```

Out[215]:

	cement	slag	ash	water	superplastic	coarseagg	fineagg	age

count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000
mean	281.167864	73.825372	54.188350	181.564658	6.145607	972.918932	773.580485	45.662136
std	104.506364	86.058467	63.997004	21.346259	5.759096	77.753954	80.175980	63.169912
min	102.000000	0.000000	0.000000	121.800000	0.000000	801.000000	594.000000	1.000000
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.950000	7.000000
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968.000000	779.500000	28.000000
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.400000	824.000000	56.000000
max	540.000000	332.608170	200.100000	245.598831	24.117482	1145.000000	992.600000	365.000000

Model

In [194…
```
X=df.drop(['strength'],axis=1)
y=df['strength']
```

In [195…
```
reg=setup(data=df,target='strength')
compare_models()
```

	Description	Value
0	Session id	8702
1	Target	strength
2	Target type	Regression
3	Data shape	(1030, 9)
4	Train data shape	(721, 9)
5	Test data shape	(309, 9)
6	Numeric features	8
7	Preprocess	True
8	Imputation type	simple
9	Numeric imputation	mean
10	Categorical imputation	mode
11	Fold Generator	KFold
12	Fold Number	10
13	CPU Jobs	-1
14	Use GPU	False
15	Log Experiment	False
16	Experiment Name	reg-default-name
17	USI	9a69

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
xgboost	Extreme Gradient Boosting	3.0960	21.0102	4.5360	0.9211	0.1447	0.1056	0.1970
lightgbm	Light Gradient Boosting Machine	3.2545	21.4994	4.5931	0.9191	0.1429	0.1097	0.2640
et	Extra Trees Regressor	3.1589	22.8870	4.7422	0.9142	0.1420	0.1055	0.1280

rf	Random Forest Regressor	3.5961	26.2568	5.0801	0.9014	0.1587	0.1242	0.1530
gbr	Gradient Boosting Regressor	3.8093	26.7742	5.1228	0.8999	0.1591	0.1289	0.0690
dt	Decision Tree Regressor	4.6383	49.0019	6.9388	0.8142	0.2171	0.1579	0.0240
ada	AdaBoost Regressor	6.4314	61.1754	7.8029	0.7706	0.2774	0.2608	0.0670
knn	K Neighbors Regressor	7.1678	93.6860	9.6137	0.6499	0.3100	0.2733	0.0290
en	Elastic Net	8.1255	106.1419	10.2470	0.6035	0.3279	0.3085	0.0270
ridge	Ridge Regression	8.1222	106.1263	10.2458	0.6035	0.3275	0.3079	0.0270
lr	Linear Regression	8.1222	106.1264	10.2458	0.6035	0.3275	0.3079	0.9690
lasso	Lasso Regression	8.1286	106.1934	10.2497	0.6033	0.3282	0.3090	0.0280
br	Bayesian Ridge	8.1315	106.4207	10.2613	0.6025	0.3292	0.3098	0.0230
huber	Huber Regressor	8.1470	109.6101	10.4274	0.5894	0.3228	0.3022	0.0420
par	Passive Aggressive Regressor	10.1858	161.5249	12.4620	0.3933	0.4166	0.3556	0.0250
lar	Least Angle Regression	9.9100	162.7940	12.5546	0.3852	0.4073	0.3647	0.0290
omp	Orthogonal Matching Pursuit	11.5913	201.8307	14.1565	0.2467	0.4575	0.4803	0.0240
llar	Lasso Least Angle Regression	13.2986	270.7701	16.4400	-0.0141	0.5241	0.5844	0.0270
dummy	Dummy Regressor	13.2986	270.7701	16.4400	-0.0141	0.5241	0.5844	0.0260

```
Processing:   0%|          | 0/81 [00:00<?, ?it/s]
```

Out[195]:

```
XGBRegressor(base_score=None, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=-1, num_parallel_tree=None,
             predictor=None, random_state=8702, ...)
```

Lightgbm

In [196…

```
from sklearn.model_selection import train_test_split
import lightgbm as lgb
```

In [197…

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

In [198…

```
lgbm=lgb.LGBMRegressor()
lgbm.fit(X_train,y_train)
```

Out[198]:

```
LGBMRegressor()
```

In [199…

```
lgbm.score(X_test,y_test)
```

Out[199]:

```
0.918382974239084
```

In [200…

```
lgbm.score(X_train,y_train)
```

Out[200]:

```
0.9813716964133293
```

In [201…
```
X_train
```

Out[201]:

	cement	slag	ash	water	superplastic	coarseagg	fineagg	age
995	380.0	0.0	0.0	228.0	0.0	932.0	670.0	365
507	251.8	0.0	99.9	146.1	12.4	1006.0	899.8	28
334	323.7	282.8	0.0	183.8	10.3	942.7	659.9	3
848	252.3	0.0	98.8	146.3	14.2	987.8	889.0	56
294	238.2	158.8	0.0	185.7	0.0	1040.6	734.3	28
...	...	...	...	...	...	...	...	...
87	212.5	0.0	100.4	159.3	8.7	1007.8	903.6	14
330	167.4	129.9	128.6	175.5	7.8	1006.3	746.6	14
466	439.0	177.0	0.0	186.0	11.1	884.9	707.9	3
121	250.0	0.0	95.7	191.8	5.3	948.9	857.2	56
860	162.0	190.0	148.0	179.0	19.0	838.0	741.0	28

824 rows × 8 columns

Extract Model

In [202…
```
joblib.dump(lgbm,'c_s.joblib')
```

Out[202]:
```
['c_s.joblib']
```

In [203…
```
cs=joblib.load('c_s.joblib')
```

In [213…
```
a=X_train[2:3]
a
```

Out[213]:

	cement	slag	ash	water	superplastic	coarseagg	fineagg	age
334	323.7	282.8	0.0	183.8	10.3	942.7	659.9	3

In [214…
```
cs.predict([[323.7,282.8,0.0,183.8,10.3,942.7,659.9,3]])
```

Out[214]:
```
array([32.5437865])
```

In [210…
```
y_train[2:3]
```

Out[210]:
```
334    28.3
Name: strength, dtype: float64
```

Standration

In [121…
```
df1=df.copy()
```

In [134…
```
X=df1.drop(['strength','ash'],axis=1)
```

In [135…
```
scaler=StandardScaler()
```

In [136…
```
scaler.fit(X)
```

Out[136]:
```
StandardScaler()
```

In [137…
```
s_d=scaler.transform(X)
```

In [138…
```
y=df1['strength']
```

In [139…
```
reg1=setup(data=X,target=y)
compare_models()
```

	Description	Value
0	Session id	6234
1	Target	strength
2	Target type	Regression
3	Data shape	(1030, 8)
4	Train data shape	(721, 8)
5	Test data shape	(309, 8)
6	Numeric features	7
7	Preprocess	True
8	Imputation type	simple
9	Numeric imputation	mean
10	Categorical imputation	mode
11	Fold Generator	KFold
12	Fold Number	10
13	CPU Jobs	-1
14	Use GPU	False
15	Log Experiment	False
16	Experiment Name	reg-default-name
17	USI	c8d8

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
lightgbm	Light Gradient Boosting Machine	3.5585	27.5975	5.1750	0.8960	0.1593	0.1224	0.0450
et	Extra Trees Regressor	3.4568	29.4587	5.3636	0.8878	0.1635	0.1186	0.1180
gbr	Gradient Boosting Regressor	3.9763	30.9675	5.4781	0.8838	0.1720	0.1375	0.0670
xgboost	Extreme Gradient Boosting	3.5520	31.0731	5.4991	0.8815	0.1698	0.1209	0.0680
rf	Random Forest Regressor	3.8981	32.5945	5.6525	0.8770	0.1754	0.1359	0.1520
dt	Decision Tree Regressor	4.8986	60.5304	7.6538	0.7699	0.2286	0.1662	0.0280
ada	AdaBoost Regressor	6.6680	66.5895	8.1146	0.7505	0.2899	0.2756	0.0750
knn	K Neighbors Regressor	7.1211	91.3703	9.4912	0.6563	0.3102	0.2759	0.0260
br	Bayesian Ridge	8.5879	116.4277	10.7373	0.5630	0.3452	0.3324	0.0230
ridge	Ridge Regression	8.5837	116.5186	10.7441	0.5627	0.3439	0.3307	0.0280

en	Elastic Net	8.5880	116.5095	10.7425	0.5627	0.3446	0.3316	0.0220
lr	Linear Regression	8.5837	116.5187	10.7441	0.5627	0.3439	0.3307	0.0790
lar	Least Angle Regression	8.5837	116.5187	10.7441	0.5627	0.3439	0.3307	0.0310
lasso	Lasso Regression	8.5936	116.5675	10.7446	0.5625	0.3451	0.3322	0.0240
huber	Huber Regressor	8.6722	124.3654	11.1040	0.5347	0.3408	0.3260	0.0350
par	Passive Aggressive Regressor	10.3783	167.0260	12.7322	0.3592	0.4116	0.4024	0.0310
omp	Orthogonal Matching Pursuit	11.7288	206.6712	14.3445	0.2238	0.4717	0.4978	0.0180
llar	Lasso Least Angle Regression	13.2627	271.7688	16.4592	-0.0162	0.5345	0.5923	0.0270
dummy	Dummy Regressor	13.2627	271.7688	16.4592	-0.0162	0.5345	0.5923	0.0260

```
Processing:   0%|          | 0/81 [00:00<?, ?it/s]
```

Out[139]:

```
LGBMRegressor(random_state=6234)
```

In [142…

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
lgbm1=lgb.LGBMRegressor()
lgbm1.fit(X_train,y_train)
lgbm1.score(X_test,y_test)
```

Out[142]:

```
0.9160112028208413
```

In [143…

```
lgbm1.score(X_train,y_train)
```

Out[143]:

```
0.9811978387851121
```

In [ ]: