Notebook Overview

Steps:

- Business Understanding.
- Analysis of Data.
- Preprocessing.
- Build model.
- Evaluate.
- Communicate.

1. Business Understanding

Breast cancer is the most common cancer amongst women in the world. It accounts for 25% of all cancer cases, and affected over 2.1 Million people in 2015 alone. It starts when cells in the breast begin to grow out of control. These cells usually form tumors that can be seen via X-ray or felt as lumps in the breast area.

2. Prepare Data

Import

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns
import scipy

import chart_studio.plotly as py
import plotly.graph_objs as go
```

```python
import plotly.graph_objs as go
import plotly.express as px

from plotly.offline import iplot, init_notebook_mode
init_notebook_mode(connected=True)

from IPython.core.display import HTML
pd.options.display.max_rows = 30
pd.options.display.max_columns = 25

import cufflinks as cf
cf.go_offline(connected=True)
cf.set_config_file(colorscale='plotly', world_readable=True)

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = 'all'

from ipywidgets import interact, interact_manual, widgets


from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split , GridSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score, precision_score, accuracy_score, recall_score, balanced_accuracy_score , ConfusionMatrixDi
```

In [2]:
```python
df = pd.read_csv('breast-cancer.csv')

df.head()
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symm
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	

0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	

5 rows × 32 columns

Explore & Analyse

In [11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
```

```
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

Dataset Description:

The dataset contains information from 32 columns, comprising 31 numerical columns and 1 categorical column.

Numerical Columns:

- Radius
- Texture
- Perimeter
- Area
- Smoothness
- Compactness
- Concavity
- Concave Points
- Symmetry
- Fractal Dimension

These columns are further categorized into three groups:

These columns are further categorized into three groups:

- **10 columns** for mean measurements.
- **10 columns** for standard error measurements.
- **10 columns** for worst measurements.

Target Variable:

classify tumors as either benign or malignant.

In [3]:
```
# Drop ID column

df.drop('id', axis=1, inplace=True)
```

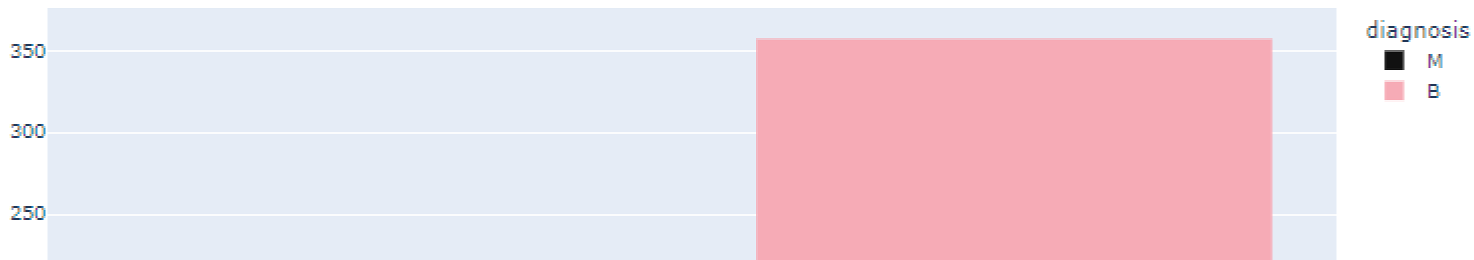
In [32]:
```
df.describe().T.style.background_gradient(cmap = sns.color_palette("ch:s=-.0,r=.6", as_cmap=True))
```

Out[32]:

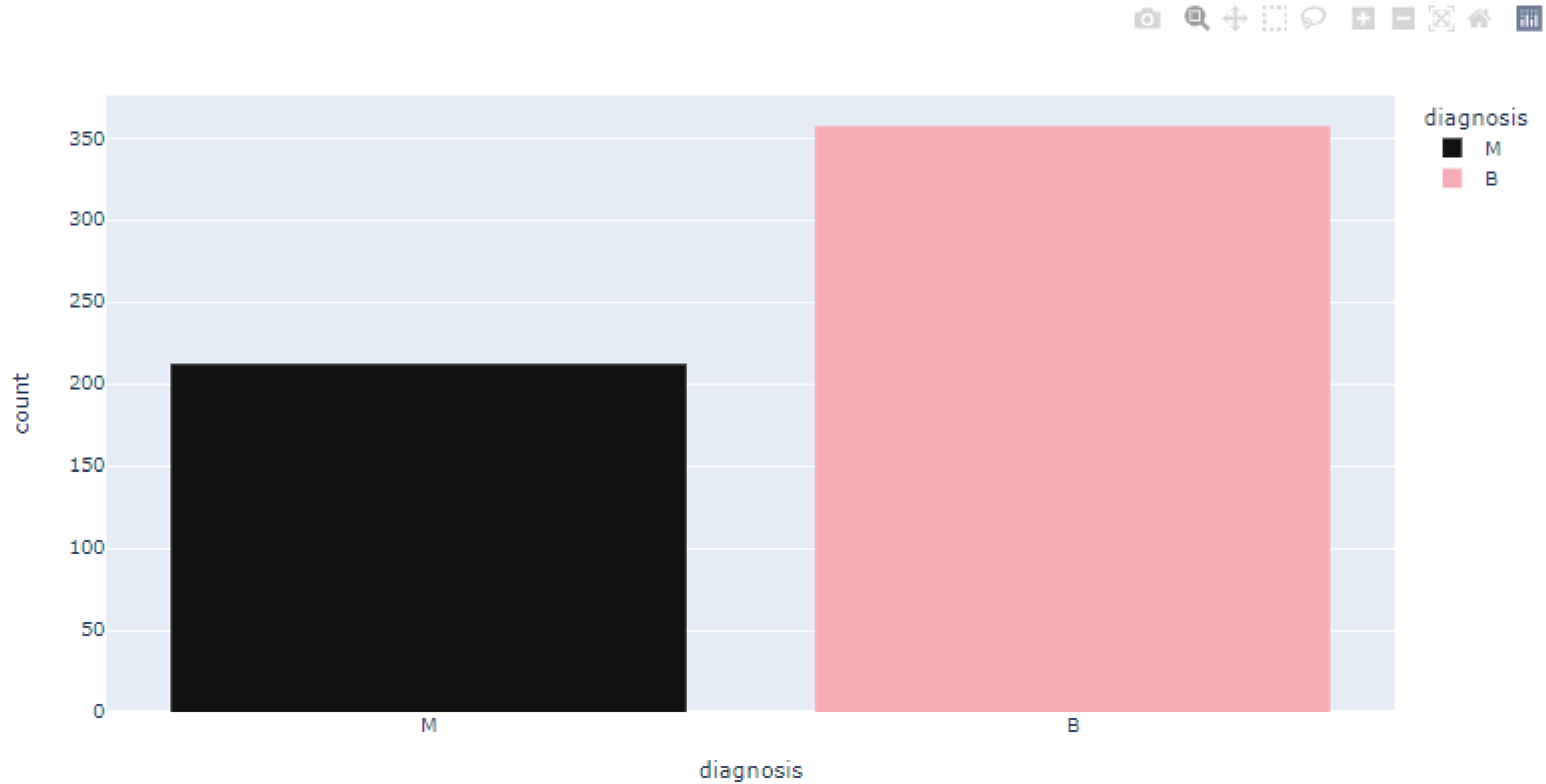
	count	mean	std	min	25%	50%	75%	max
radius_mean	569.000000	14.127292	3.524049	6.981000	11.700000	13.370000	15.780000	28.110000
texture_mean	569.000000	19.289649	4.301036	9.710000	16.170000	18.840000	21.800000	39.280000
perimeter_mean	569.000000	91.969033	24.298981	43.790000	75.170000	86.240000	104.100000	188.500000
area_mean	569.000000	654.889104	351.914129	143.500000	420.300000	551.100000	782.700000	2501.000000
smoothness_mean	569.000000	0.096360	0.014064	0.052630	0.086370	0.095870	0.105300	0.163400
compactness_mean	569.000000	0.104341	0.052813	0.019380	0.064920	0.092630	0.130400	0.345400
concavity_mean	569.000000	0.088799	0.079720	0.000000	0.029560	0.061540	0.130700	0.426800
concave points_mean	569.000000	0.048919	0.038803	0.000000	0.020310	0.033500	0.074000	0.201200
symmetry_mean	569.000000	0.181162	0.027414	0.106000	0.161900	0.179200	0.195700	0.304000
fractal_dimension_mean	569.000000	0.062798	0.007060	0.049960	0.057700	0.061540	0.066120	0.097440
radius_se	569.000000	0.405172	0.277313	0.111500	0.232400	0.324200	0.478900	2.873000
texture_se	569.000000	1.216853	0.551648	0.360200	0.833900	1.108000	1.474000	4.885000
perimeter_se	569.000000	2.866059	2.021855	0.757000	1.606000	2.287000	3.357000	21.980000
area_se	569.000000	40.337079	45.491006	6.802000	17.850000	24.530000	45.190000	542.200000

area_se	569.000000	40.337079	45.491006	6.802000	17.850000	24.530000	45.190000	542.200000
smoothness_se	569.000000	0.007041	0.003003	0.001713	0.005169	0.006380	0.008146	0.031130
compactness_se	569.000000	0.025478	0.017908	0.002252	0.013080	0.020450	0.032450	0.135400
concavity_se	569.000000	0.031894	0.030186	0.000000	0.015090	0.025890	0.042050	0.396000
concave points_se	569.000000	0.011796	0.006170	0.000000	0.007638	0.010930	0.014710	0.052790
symmetry_se	569.000000	0.020542	0.008266	0.007882	0.015160	0.018730	0.023480	0.078950
fractal_dimension_se	569.000000	0.003795	0.002646	0.000895	0.002248	0.003187	0.004558	0.029840
radius_worst	569.000000	16.269190	4.833242	7.930000	13.010000	14.970000	18.790000	36.040000
texture_worst	569.000000	25.677223	6.146258	12.020000	21.080000	25.410000	29.720000	49.540000
perimeter_worst	569.000000	107.261213	33.602542	50.410000	84.110000	97.660000	125.400000	251.200000
area_worst	569.000000	880.583128	569.356993	185.200000	515.300000	686.500000	1084.000000	4254.000000
smoothness_worst	569.000000	0.132369	0.022832	0.071170	0.116600	0.131300	0.146000	0.222600
compactness_worst	569.000000	0.254265	0.157336	0.027290	0.147200	0.211900	0.339100	1.058000
concavity_worst	569.000000	0.272188	0.208624	0.000000	0.114500	0.226700	0.382900	1.252000
concave points_worst	569.000000	0.114606	0.065732	0.000000	0.064930	0.099930	0.161400	0.291000
symmetry_worst	569.000000	0.290076	0.061867	0.156500	0.250400	0.282200	0.317900	0.663800
fractal_dimension_worst	569.000000	0.083946	0.018061	0.055040	0.071460	0.080040	0.092080	0.207500

In [33]:
```
px.histogram(data_frame=df, x='diagnosis', color='diagnosis',color_discrete_sequence=['#111111','#f6abb6'])
```



In [33]:
```
px.histogram(data_frame=df, x='diagnosis', color='diagnosis',color_discrete_sequence=['#111111','#f6abb6'])
```

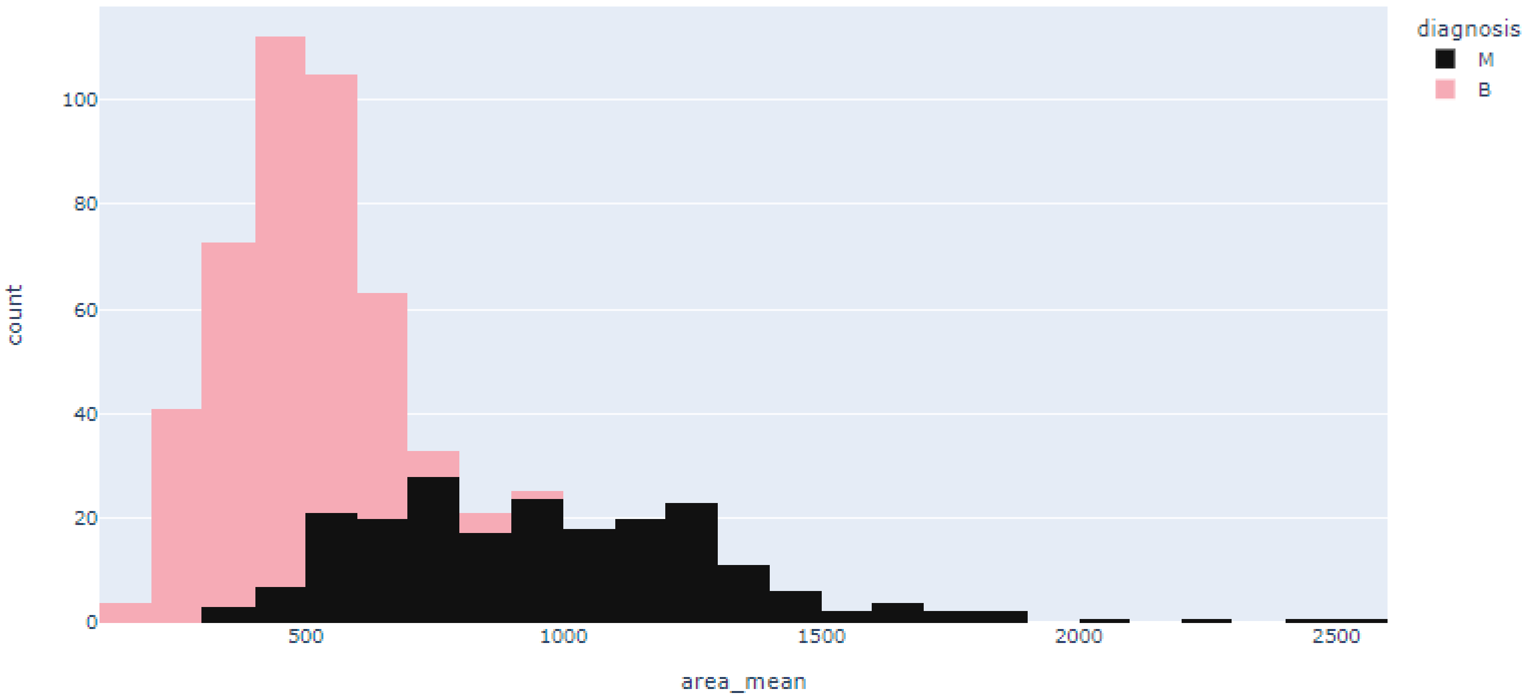


In [13]:
```
df['diagnosis'].value_counts(normalize=True)
```

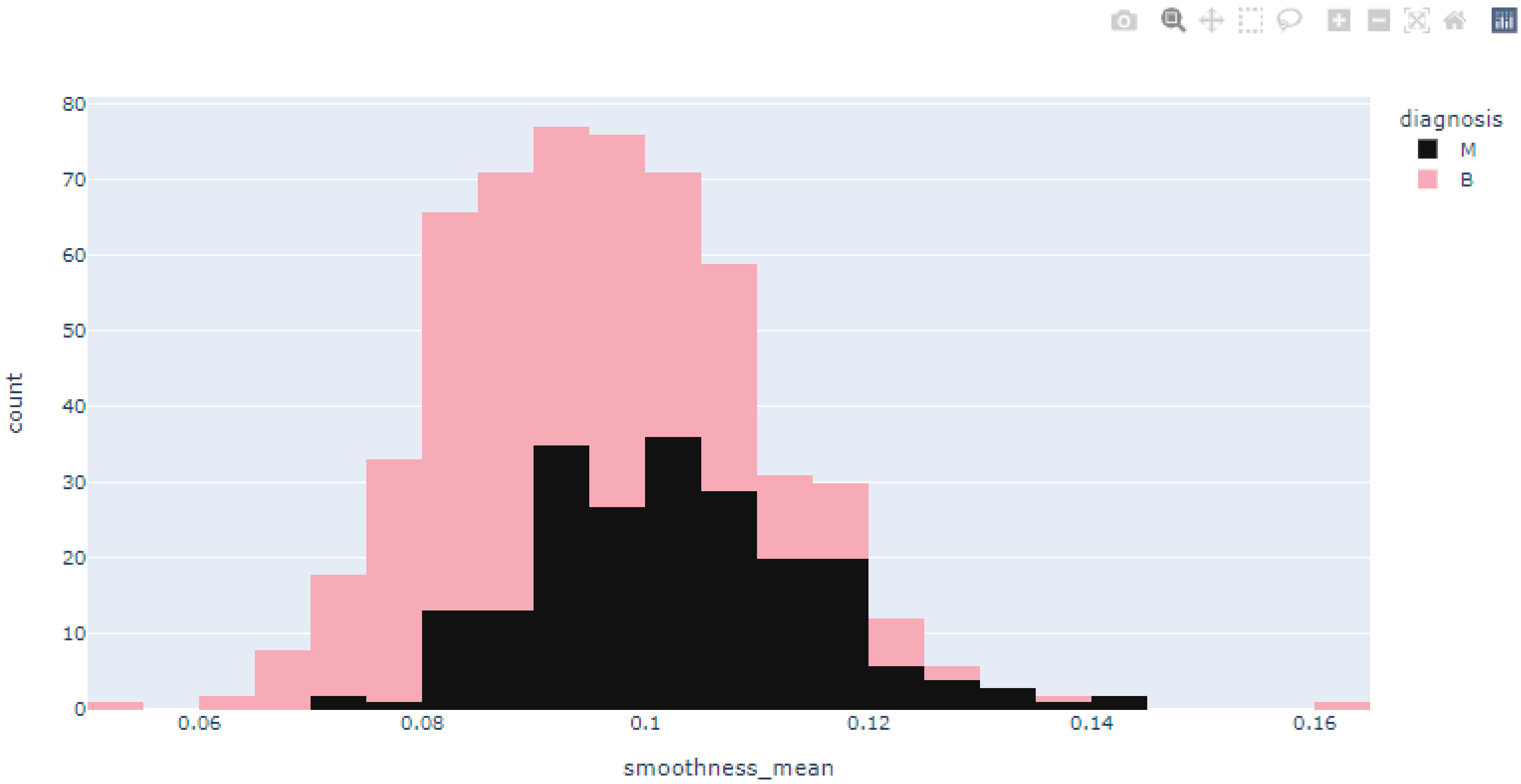
Out[13]:
```
B    0.627417
M    0.372583
Name: diagnosis, dtype: float64
```

It shows us that we have an imbalanced dataset, where the benign class is 63%, compared to the malignant class, which is 37%

In [35]:
```
px.histogram(data_frame=df,x=df.area_mean,color='diagnosis',color_discrete_sequence=['#111111','#f6abb6'])
```



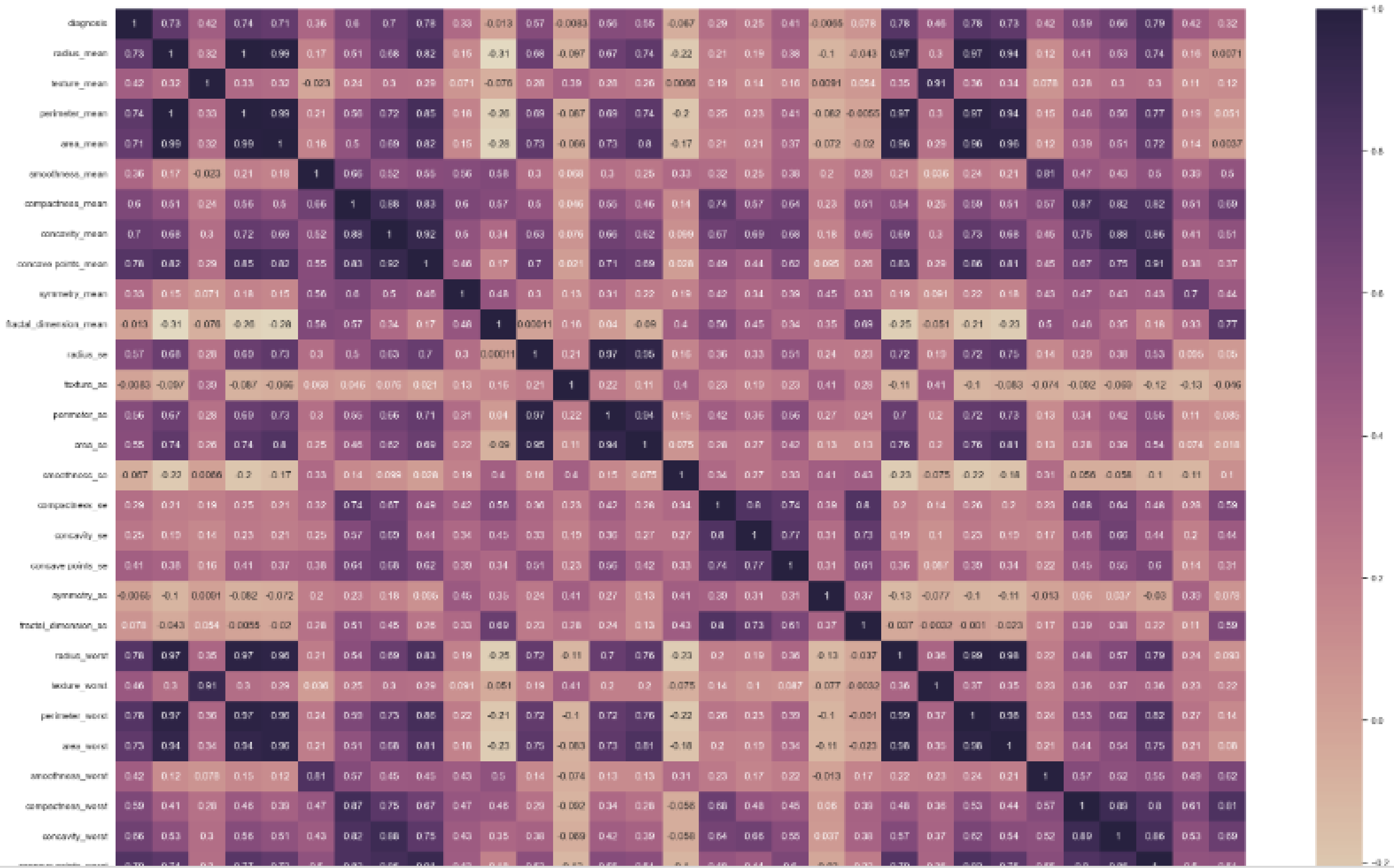
In [36]:
```python
px.histogram(data_frame=df,x='smoothness_mean',color='diagnosis',color_discrete_sequence=['#111111','#f6abb6'])
```

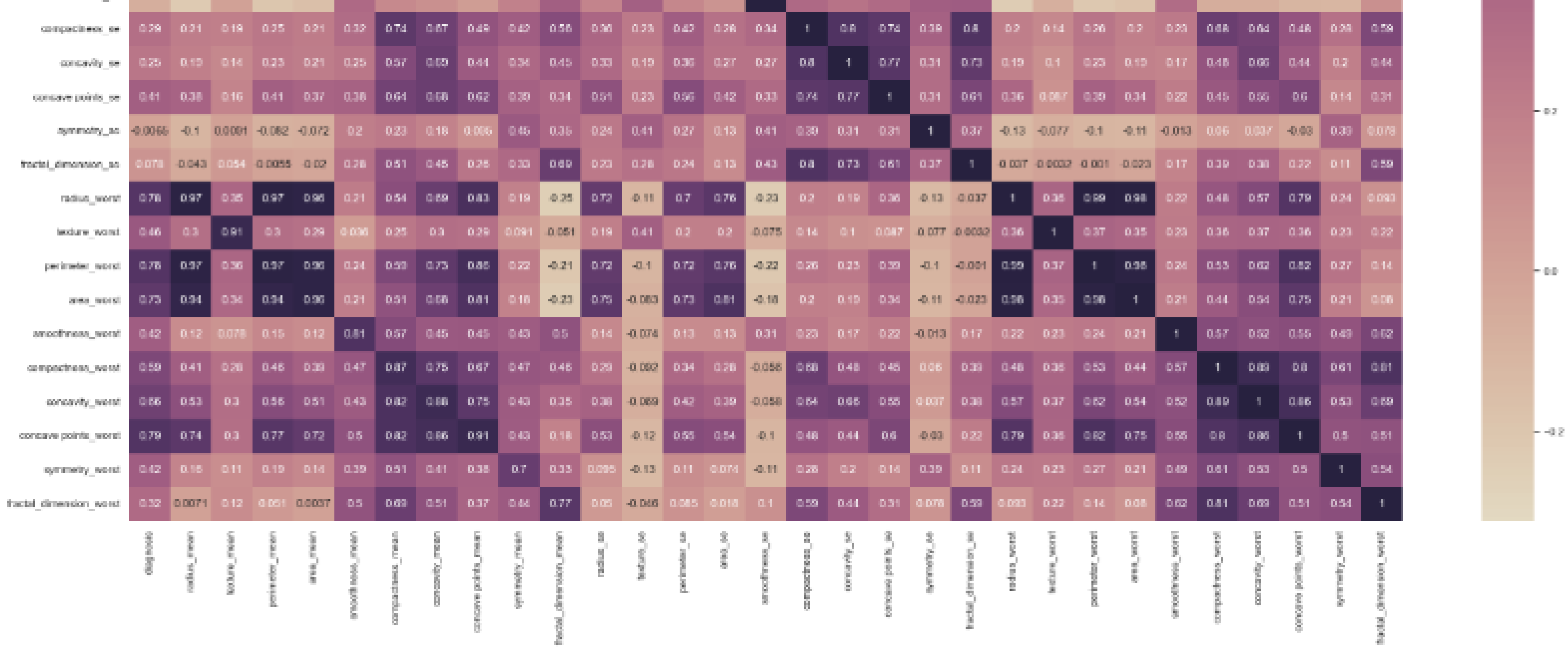


In [4]:
```python
# Convert the categorical data into numerical as (Malignant:1 ,Benign:0)
df['diagnosis'] = (df['diagnosis'] == 'M').astype(int)
```

In [38]:

```python
# Create a heatmap
plt.figure(figsize=(30,20))
sns.heatmap(df.corr(), annot=True, cmap=sns.color_palette("ch:s=-.2,r=.6", as_cmap=True))
plt.show();
```



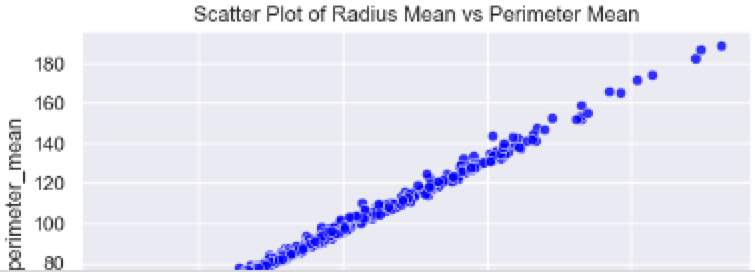


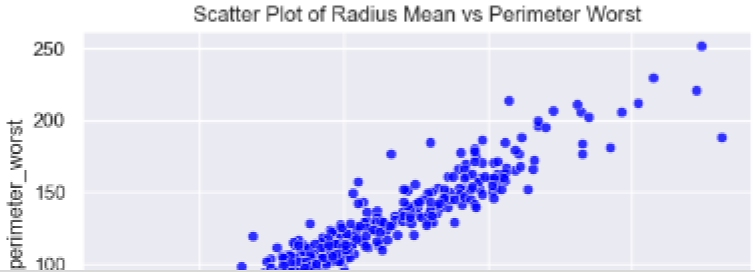
Here are some scatterplots depicting positive linear regression, extracted from the heatmap analysis :

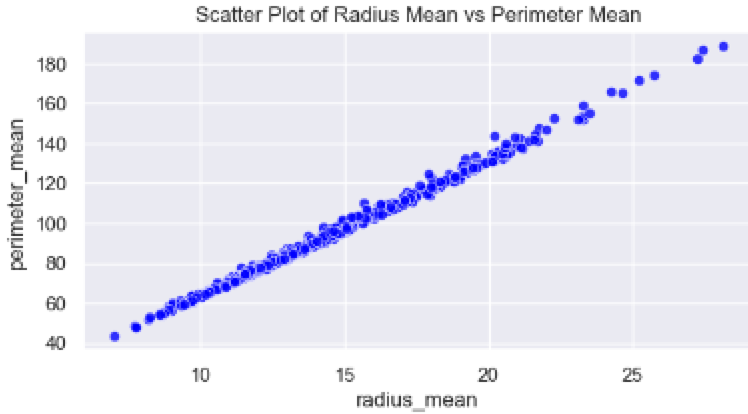
In [39]:
```
sns.set(style="darkgrid")
fig, axs = plt.subplots(3, 2, figsize=(15, 15))

plt.subplots_adjust(hspace=0.8)

# Plot the first scatterplot
sns.scatterplot(x='radius_mean', y='perimeter_mean', data=df, color='blue', alpha=0.8 , ax=axs[0, 0])
axs[0, 0].set_title('Scatter Plot of Radius Mean vs Perimeter Mean')

# Plot the second scatterplot
sns.scatterplot(x='radius_mean', y='perimeter_worst', data=df, color='blue', alpha=0.8, ax=axs[0, 1])
axs[0, 1].set_title('Scatter Plot of Radius Mean vs Perimeter Worst')
```

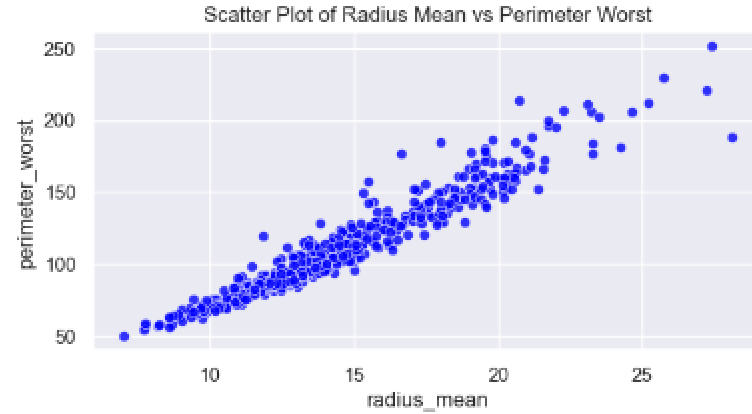
In [39]:

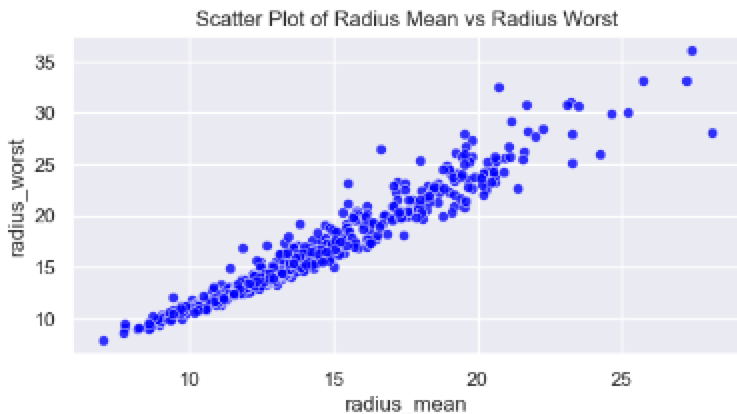
```python
sns.set(style="darkgrid")
fig, axs = plt.subplots(3, 2, figsize=(15, 15))

plt.subplots_adjust(hspace=0.8)

# Plot the first scatterplot
sns.scatterplot(x='radius_mean', y='perimeter_mean', data=df, color='blue', alpha=0.8 , ax=axs[0, 0])
axs[0, 0].set_title('Scatter Plot of Radius Mean vs Perimeter Mean')

# Plot the second scatterplot
sns.scatterplot(x='radius_mean', y='perimeter_worst', data=df, color='blue', alpha=0.8, ax=axs[0, 1])
axs[0, 1].set_title('Scatter Plot of Radius Mean vs Perimeter Worst')

# Plot the third scatterplot
sns.scatterplot(x='radius_mean', y='radius_worst', data=df, color='blue', alpha=0.8, ax=axs[1, 0])
axs[1, 0].set_title('Scatter Plot of Radius Mean vs Radius Worst')

# Plot the fourth scatterplot
sns.scatterplot(x='radius_mean', y='area_mean', data=df, color='blue', alpha=0.8, ax=axs[1, 1])
axs[1, 1].set_title('Scatter Plot of Radius Mean vs Area Mean')

# Plot the fifth scatterplot
sns.scatterplot(x='radius_mean', y='perimeter_mean', data=df, color='blue', alpha=0.8, ax=axs[2, 0])
axs[2, 0].set_title('Scatter Plot of Radius Mean vs Perimeter Mean')

# Plot the sixth scatterplot
sns.scatterplot(x='perimeter_se', y='area_se', data=df, color='blue', alpha=0.8)
axs[2, 1].set_title('Scatter Plot of Perimeter Se vs Area Se')

plt.show();
```

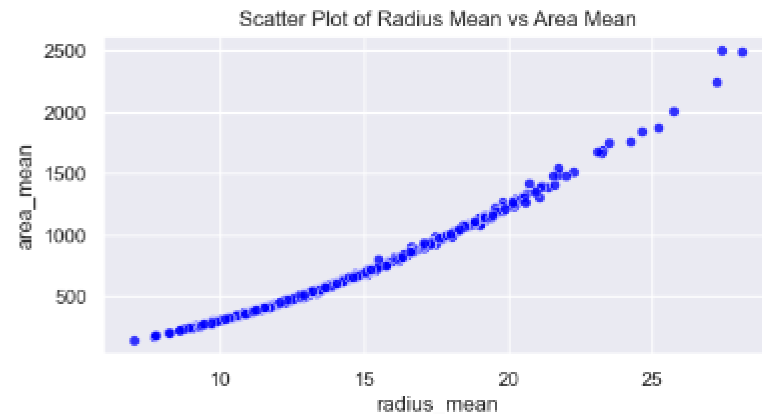


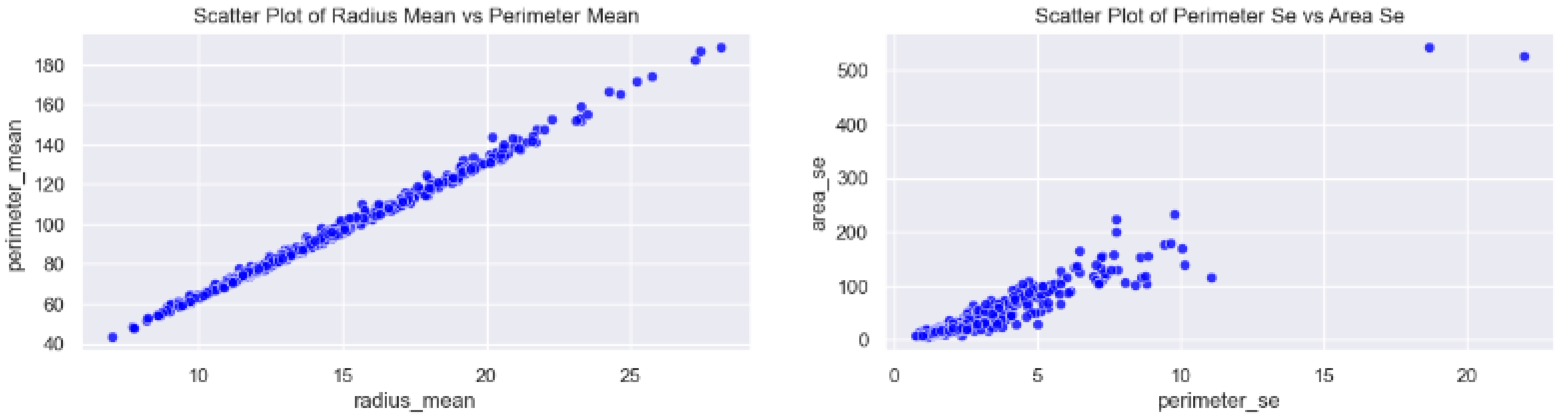



Scatter Plot of Radius Mean vs Perimeter Mean
perimeter_mean
radius_mean
40
60
80
100
120
140
160
180
10
15
20
25


Scatter Plot of Radius Mean vs Perimeter Worst
perimeter_worst
radius_mean
50
100
150
200
250
10
15
20
25


Scatter Plot of Radius Mean vs Radius Worst
radius_worst
radius_mean
10
15
20
25
30
35
10
15
20
25


Scatter Plot of Radius Mean vs Area Mean
area_mean
radius_mean
500
1000
1500
2000
2500
10
15
20
25



we construct a widget that quickly finds the correlation between two columns in the dataset.

In [5]:
```
@interact
def correlations(column1=list(df.select_dtypes('number').columns),
                 column2=list(df.select_dtypes('number').columns)):
    print(f"Correlation: {df[column1].corr(df[column2])}")
```

column1 perimeter_se

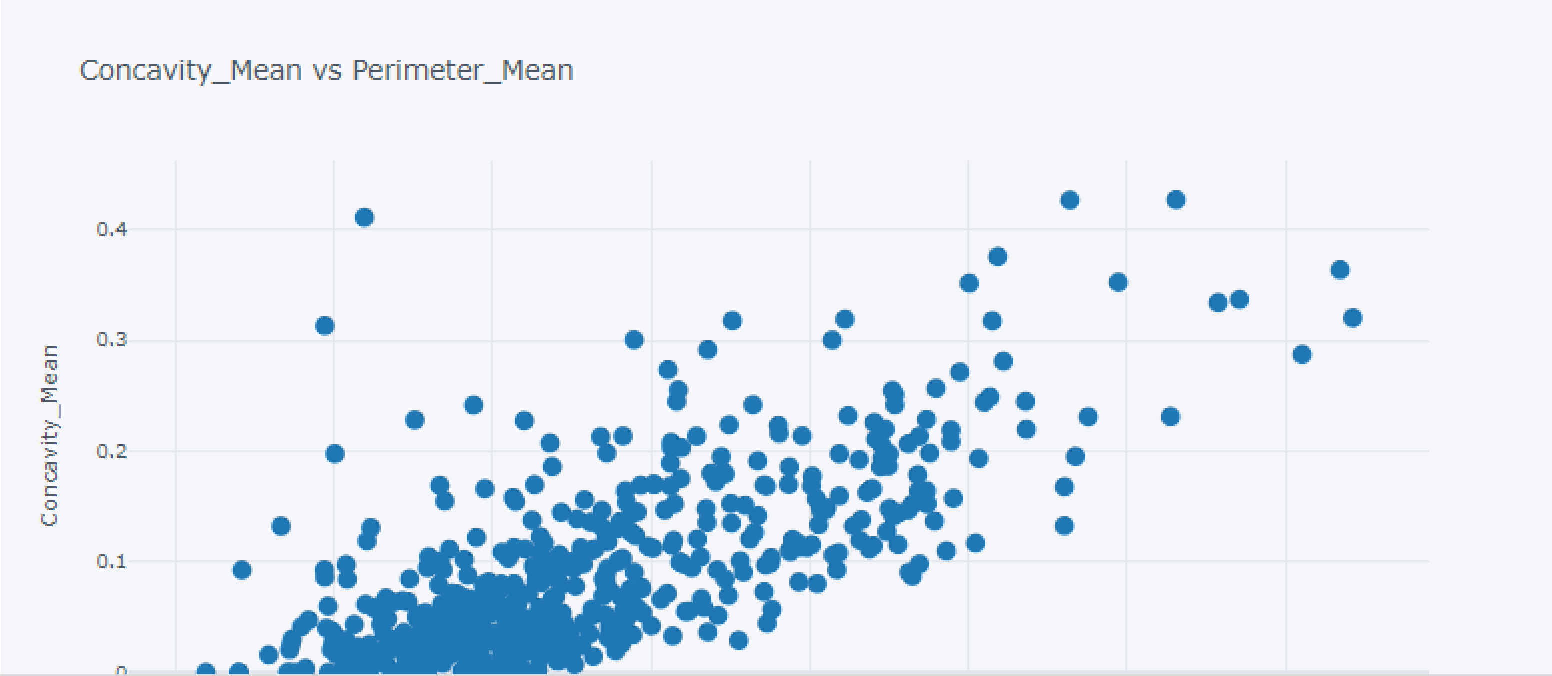
column2 area_se

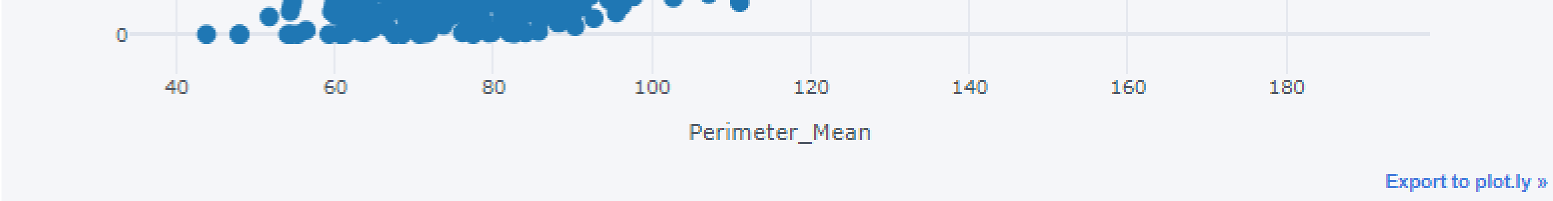
```
Correlation: 0.9376554069544155
```

In [6]:
```
@interact
def scatter_plot(x=list(df.select_dtypes('number').columns),
                 y=list(df.select_dtypes('number').columns)[1:]):
    if x == y:
        print(f"Please select separate variables for X and Y")
    else:
        df.iplot(kind='scatter', x=x, y=y, mode='markers',
                 xTitle=x.title(), yTitle=y.title(), title=f'{y.title()} vs {x.title()}')
```

x perimeter_mean

y concavity_mean

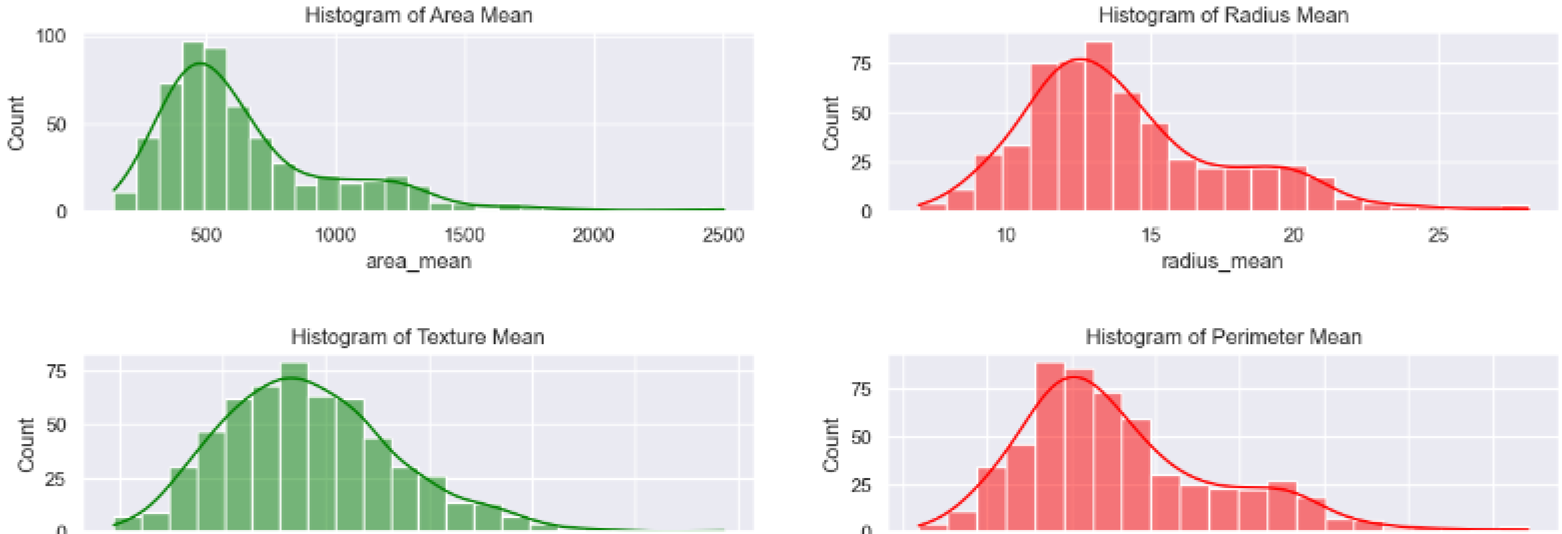


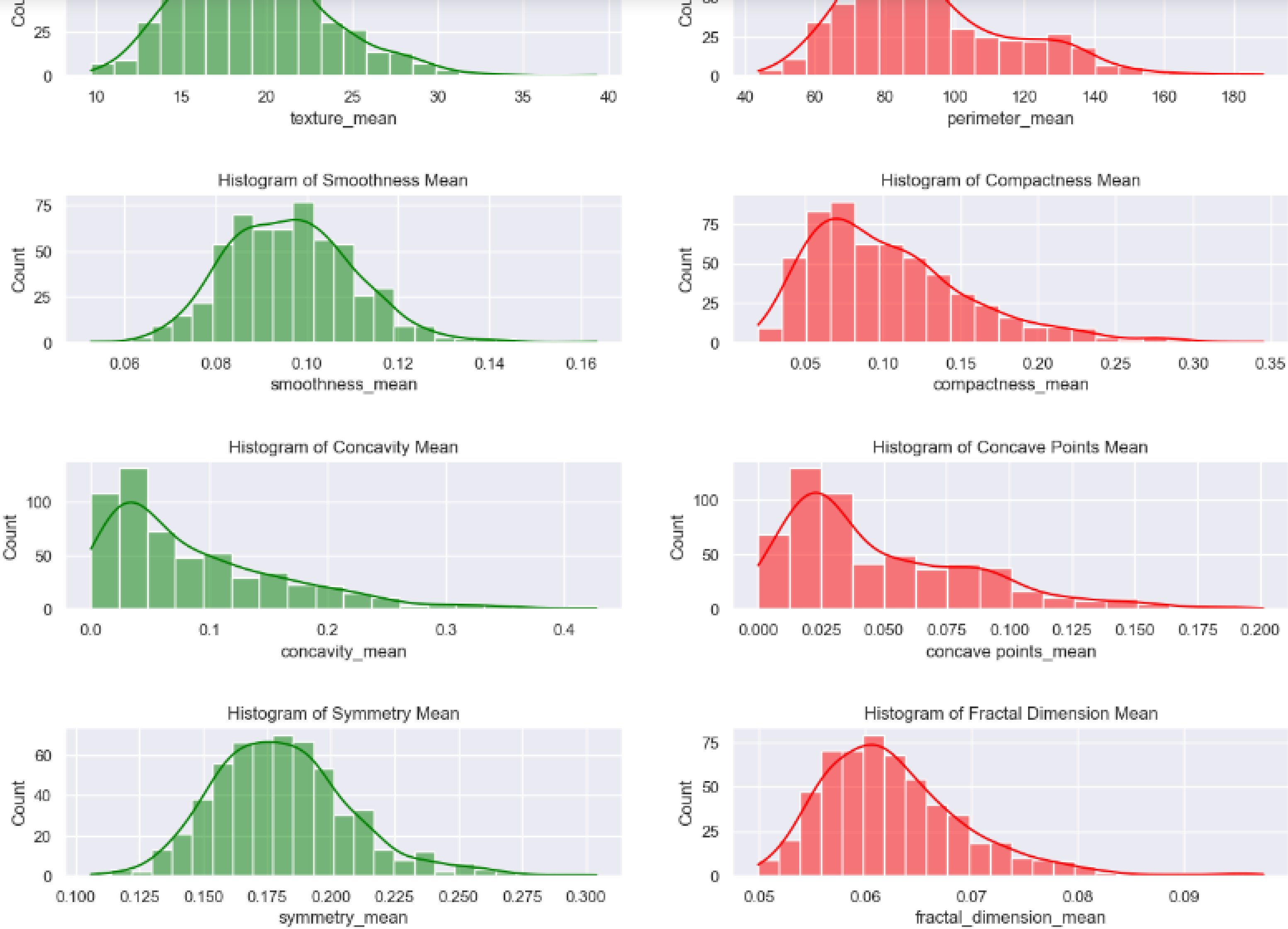


In [42]:

```
# Set the style
sns.set(style="darkgrid")

# Create two separate subplots
fig, axs = plt.subplots(5, 2, figsize=(15, 15))  # Increase the figure size

# Adjust the spacing between subplots
plt.subplots_adjust(hspace=0.8)  # Increase the vertical spacing

# Plot the first histogram
sns.histplot(data=df, x="area_mean", kde=True, ax=axs[0, 0], color='green')
axs[0, 0].set_title('Histogram of Area Mean')

# Plot the second histogram
sns.histplot(data=df, x="radius_mean", kde=True, ax=axs[0, 1], color='red')
axs[0, 1].set_title('Histogram of Radius Mean')

# Plot the third histogram
sns.histplot(data=df, x="texture_mean", kde=True, ax=axs[1, 0], color='green')
axs[1, 0].set_title('Histogram of Texture Mean')

# Plot the fourth histogram
sns.histplot(data=df, x="perimeter_mean", kde=True, ax=axs[1, 1], color='red')
axs[1, 1].set_title('Histogram of Perimeter Mean')

# Plot the fifth histogram
sns.histplot(data=df, x="smoothness_mean", kde=True, ax=axs[2, 0], color='green')
axs[2, 0].set_title('Histogram of Smoothness Mean')

# Plot the sixth histogram
```

```
# Plot the sixth histogram
sns.histplot(data=df, x="compactness_mean", kde=True, ax=axs[2, 1], color='red')
axs[2, 1].set_title('Histogram of Compactness Mean')

# Plot the seventh histogram
sns.histplot(data=df, x="concavity_mean", kde=True, ax=axs[3, 0], color='green')
axs[3, 0].set_title('Histogram of Concavity Mean')

# Plot the eighth histogram
sns.histplot(data=df, x="concave points_mean", kde=True, ax=axs[3, 1], color='red')
axs[3, 1].set_title('Histogram of Concave Points Mean')

# Plot the ninth histogram
sns.histplot(data=df, x="symmetry_mean", kde=True, ax=axs[4, 0], color='green')
axs[4, 0].set_title('Histogram of Symmetry Mean')

# Plot the tenth histogram
sns.histplot(data=df, x="fractal_dimension_mean", kde=True, ax=axs[4, 1], color='red')
axs[4, 1].set_title('Histogram of Fractal Dimension Mean')

plt.show();
```

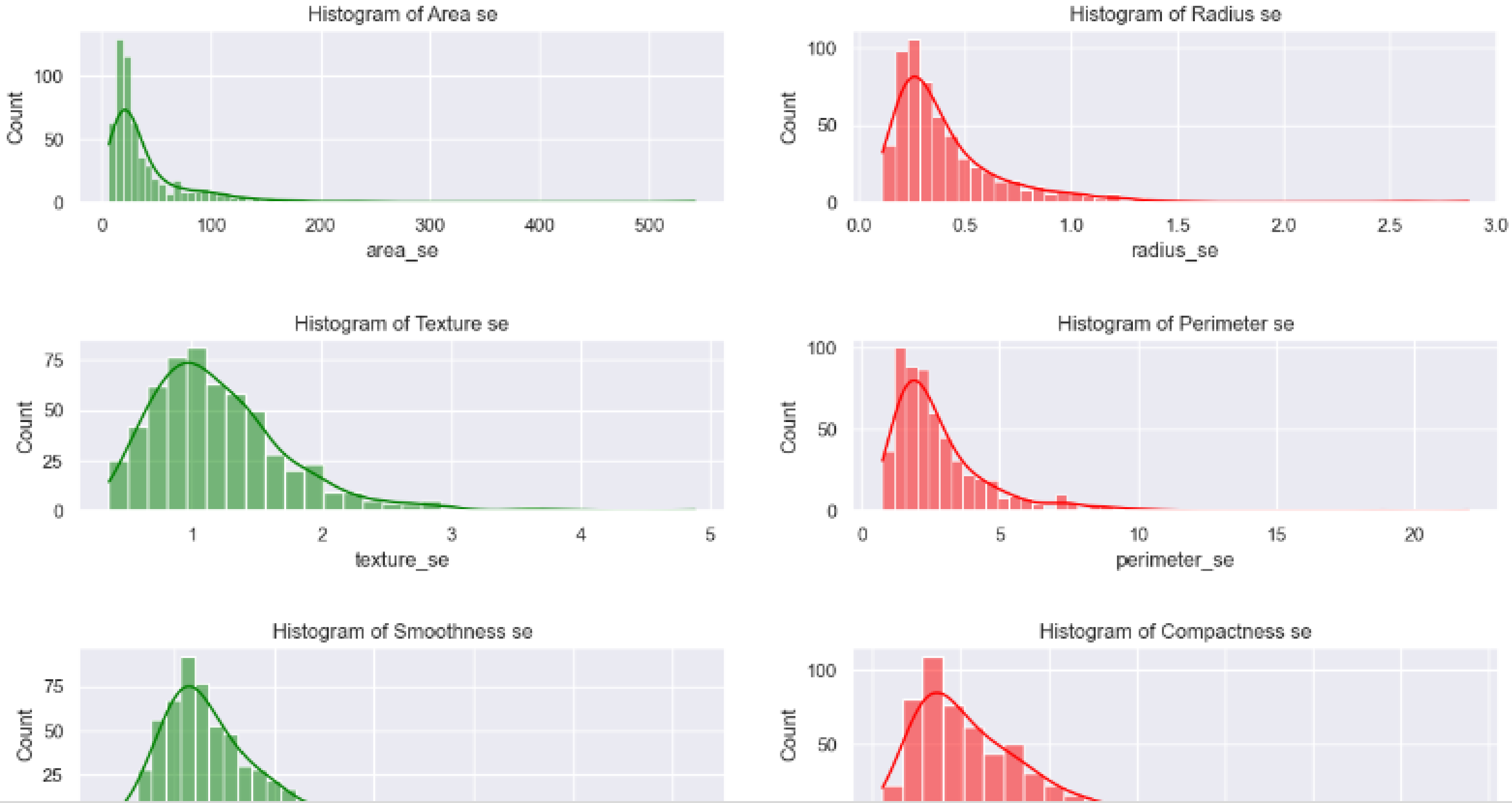


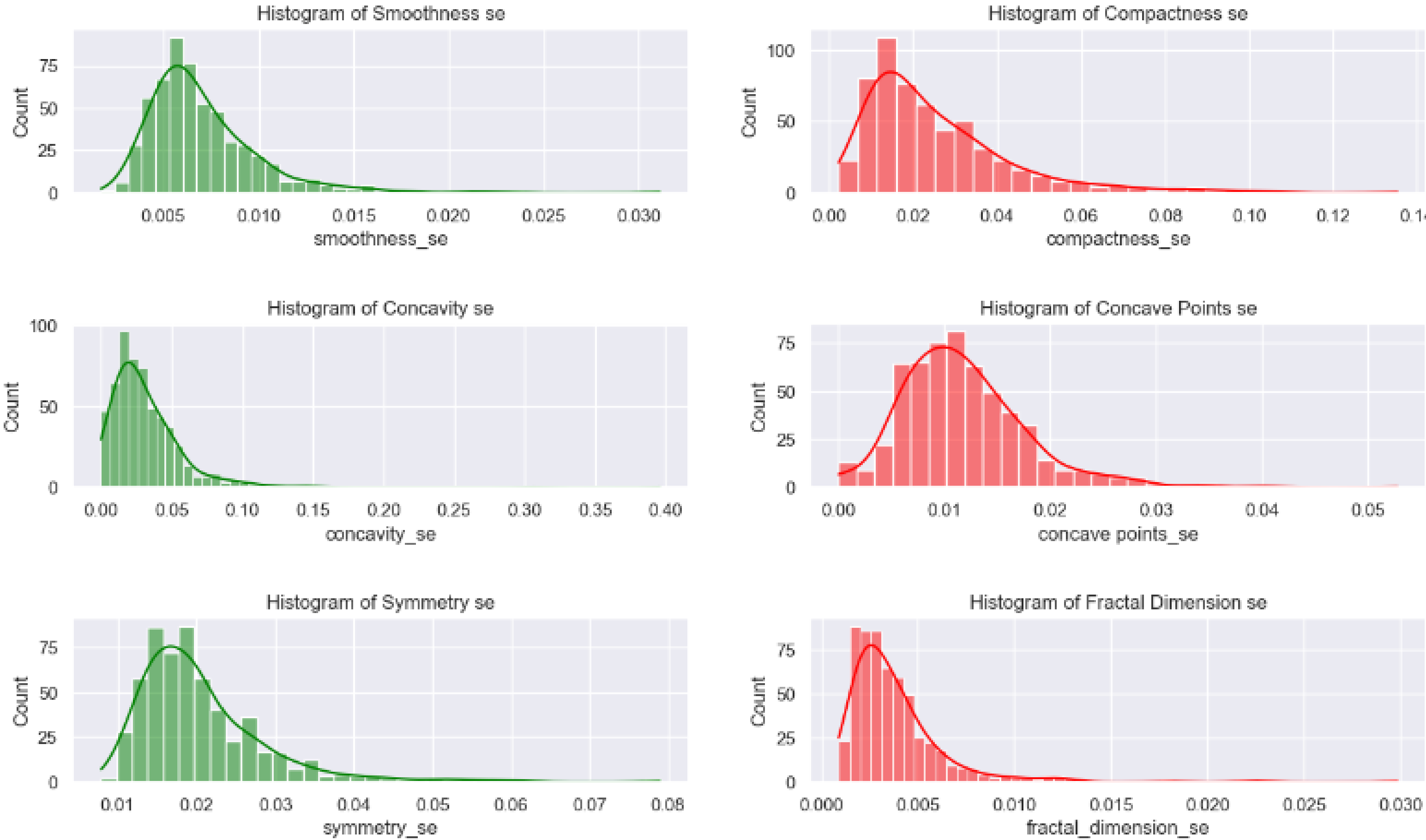

texture_mean
perimeter_mean
Histogram of Smoothness Mean
Count
smoothness_mean
Histogram of Compactness Mean
Count
compactness_mean
Histogram of Concavity Mean
Count
concavity_mean
Histogram of Concave Points Mean
Count
concave points_mean
Histogram of Symmetry Mean
Count
symmetry_mean
Histogram of Fractal Dimension Mean
Count
fractal_dimension_mean

In [43]:

```python
# Set the style
sns.set(style="darkgrid")

# Create two separate subplots
fig, axs = plt.subplots(5, 2, figsize=(15, 15))

# Adjust the spacing between subplots
plt.subplots_adjust(hspace=0.8)

# Plot the first histogram
sns.histplot(data=df, x="area_se", kde=True, ax=axs[0, 0], color='green')
axs[0, 0].set_title('Histogram of Area se')

# Plot the second histogram
sns.histplot(data=df, x="radius_se", kde=True, ax=axs[0, 1], color='red')
axs[0, 1].set_title('Histogram of Radius se')

# Plot the third histogram
sns.histplot(data=df, x="texture_se", kde=True, ax=axs[1, 0], color='green')
axs[1, 0].set_title('Histogram of Texture se')

# Plot the fourth histogram
sns.histplot(data=df, x="perimeter_se", kde=True, ax=axs[1, 1], color='red')
axs[1, 1].set_title('Histogram of Perimeter se')

# Plot the fifth histogram
sns.histplot(data=df, x="smoothness_se", kde=True, ax=axs[2, 0], color='green')
axs[2, 0].set_title('Histogram of Smoothness se')

# Plot the sixth histogram
sns.histplot(data=df, x="compactness_se", kde=True, ax=axs[2, 1], color='red')
axs[2, 1].set_title('Histogram of Compactness se')

# Plot the seventh histogram
sns.histplot(data=df, x="concavity_se", kde=True, ax=axs[3, 0], color='green')
axs[3, 0].set_title('Histogram of Concavity se')

# Plot the eighth histogram
sns.histplot(data=df, x="concave points_se", kde=True, ax=axs[3, 1], color='red')
axs[3, 1].set_title('Histogram of Concave Points se')
```

```
# Plot the ninth histogram
sns.histplot(data=df, x="symmetry_se", kde=True, ax=axs[4, 0], color='green')
axs[4, 0].set_title('Histogram of Symmetry se')

# Plot the tenth histogram
sns.histplot(data=df, x="fractal_dimension_se", kde=True, ax=axs[4, 1], color='red')
axs[4, 1].set_title('Histogram of Fractal Dimension se')

plt.show();
```





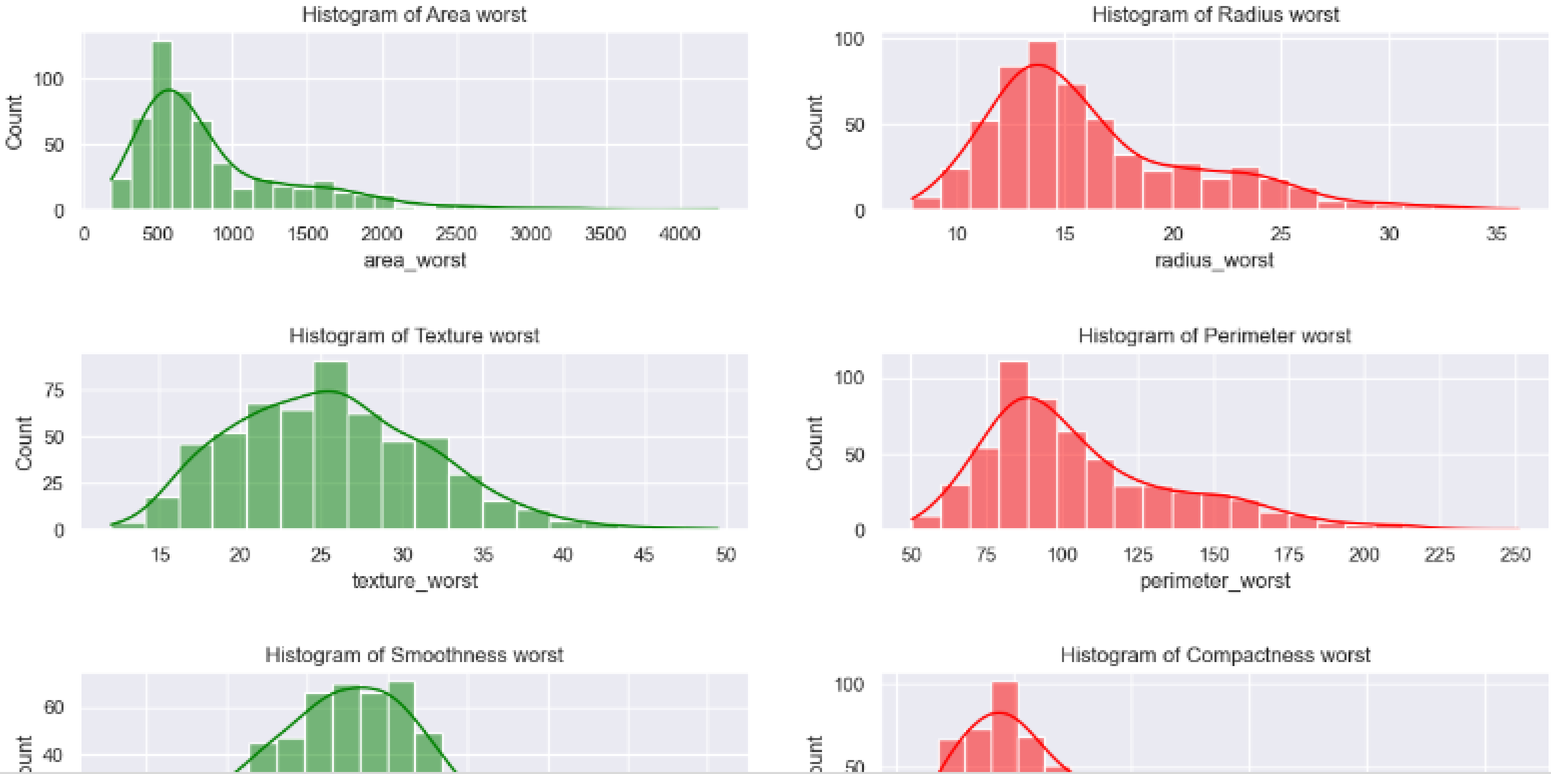
In [44]:

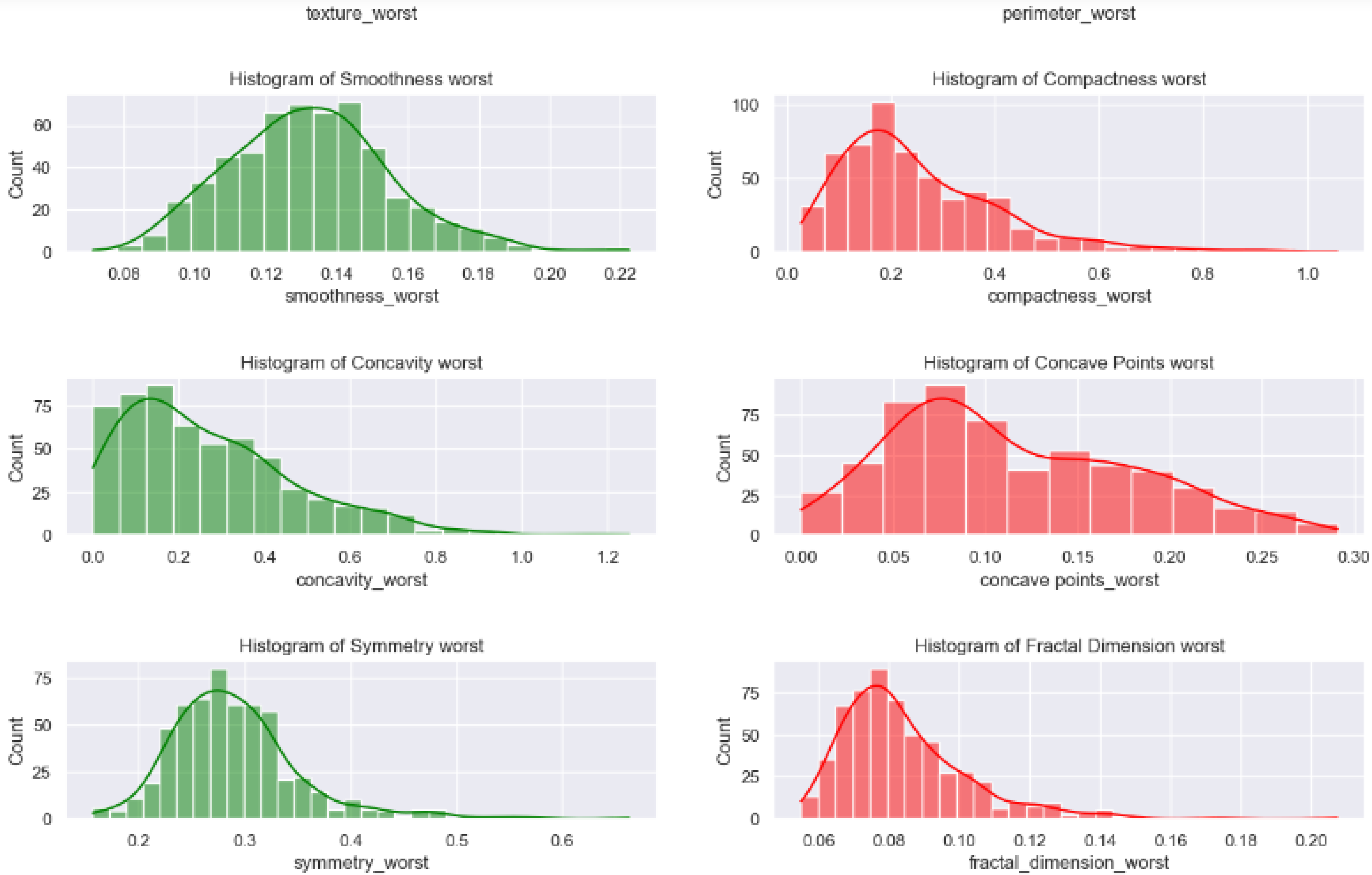
```
# Set the style
sns.set(style="darkgrid")
```

In [44]:

```python
# Set the style
sns.set(style="darkgrid")

# Create two separate subplots
fig, axs = plt.subplots(5, 2, figsize=(15, 15))

# Adjust the spacing between subplots
plt.subplots_adjust(hspace=0.8)

# Plot the first histogram
sns.histplot(data=df, x="area_worst", kde=True, ax=axs[0, 0], color='green')
axs[0, 0].set_title('Histogram of Area worst')

# Plot the second histogram
sns.histplot(data=df, x="radius_worst", kde=True, ax=axs[0, 1], color='red')
axs[0, 1].set_title('Histogram of Radius worst')

# Plot the third histogram
sns.histplot(data=df, x="texture_worst", kde=True, ax=axs[1, 0], color='green')
axs[1, 0].set_title('Histogram of Texture worst')

# Plot the fourth histogram
sns.histplot(data=df, x="perimeter_worst", kde=True, ax=axs[1, 1], color='red')
axs[1, 1].set_title('Histogram of Perimeter worst')

# Plot the fifth histogram
sns.histplot(data=df, x="smoothness_worst", kde=True, ax=axs[2, 0], color='green')
axs[2, 0].set_title('Histogram of Smoothness worst')

# Plot the sixth histogram
sns.histplot(data=df, x="compactness_worst", kde=True, ax=axs[2, 1], color='red')
axs[2, 1].set_title('Histogram of Compactness worst')

# Plot the seventh histogram
sns.histplot(data=df, x="concavity_worst", kde=True, ax=axs[3, 0], color='green')
axs[3, 0].set_title('Histogram of Concavity worst')

# Plot the eighth histogram
sns.histplot(data=df, x="concave points_worst", kde=True, ax=axs[3, 1], color='red')
axs[3, 1].set_title('Histogram of Concave Points worst')
```

```
sns.histplot(data=df, x="concave points_worst", kde=True, ax=axs[3, 1], color='red')
axs[3, 1].set_title('Histogram of Concave Points worst')

# Plot the ninth histogram
sns.histplot(data=df, x="symmetry_worst", kde=True, ax=axs[4, 0], color='green')
axs[4, 0].set_title('Histogram of Symmetry worst')

# Plot the tenth histogram
sns.histplot(data=df, x="fractal_dimension_worst", kde=True, ax=axs[4, 1], color='red')
axs[4, 1].set_title('Histogram of Fractal Dimension worst')

plt.show();
```





We conclude that most of these graphs tend to exhibit a normal distribution, while some show a tendency towards positive skewness

3. Preprocessing

Split ¶

In [45]:
```
target = 'diagnosis'
X = df.drop(columns=target)
y = df[target]

print("X shape:", X.shape)
print("y shape:", y.shape)
```

```
X shape: (569, 30)
y shape: (569,)
```

In [46]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=42)

print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (455, 30)
y_train shape: (455,)
X_test shape: (114, 30)
y_test shape: (114,)
```

In [47]:
```
scaler = StandardScaler()
scaler.fit(X_train)

scaler.transform(X_train)
scaler.transform(X_test)
```

Out[47]:
```
StandardScaler()
```

Out[47]:
```
array([[-1.44075296, -0.43531947, -1.36208497, ...,  0.9320124 ,
         2.09724217,  1.88645014],
       [ 1.97409619,  1.73302577,  2.09167167, ...,  2.6989469 ,
         1.89116053,  2.49783848],
```

```
        0.59760192,  0.0578942 ],
       ...,
       [ 0.04880192, -0.55500086, -0.06512547, ..., -1.23903365,
        -0.70863864, -1.27145475],
       [-0.03896885,  0.10207345, -0.03137406, ...,  1.05001236,
         0.43432185,  1.21336207],
       [-0.54860557,  0.31327591, -0.60350155, ..., -0.61102866,
        -0.3345212 , -0.84628745]])
```

Out[47]:
```
array([[-0.46649743, -0.13728933, -0.44421138, ..., -0.19435087,
         0.17275669,  0.20372995],
       [ 1.36536344,  0.49866473,  1.30551088, ...,  0.99177862,
        -0.561211  , -1.00838949],
       [ 0.38006578,  0.06921974,  0.40410139, ...,  0.57035018,
        -0.10783139, -0.20629287],
       ...,
       [-0.73547237, -0.99852603, -0.74138839, ..., -0.27741059,
        -0.3820785 , -0.32408328],
       [ 0.02898271,  2.0334026 ,  0.0274851 , ..., -0.49027026,
        -1.60905688, -0.33137507],
       [ 1.87216885,  2.80077153,  1.80354992, ...,  0.7925579 ,
        -0.05868885, -0.09467243]])
```

4. Build Model

In [48]:
```
def train_evaluate_model(model, X_train, y_train, X_test,y_test):

    model.fit(X_train, y_train)  #fit the model instance


    predictions = model.predict(X_test) # calculate predictions

    accuracy = accuracy_score(y_test, predictions)
    f1 = f1_score(y_test, predictions)
    precision = precision_score(y_test, predictions)
    recall = recall_score(y_test, predictions)

    #create a dataframe to visualize the results
    eval_df = pd.DataFrame([[accuracy, f1, precision, recall]], columns=['accuracy',
                                                'f1_score', 'precision', 'recall'])
```

```python
    predictions = model.predict(X_test) # calculate predictions

    accuracy = accuracy_score(y_test, predictions)
    f1 = f1_score(y_test, predictions)
    precision = precision_score(y_test, predictions)
    recall = recall_score(y_test, predictions)

    #create a dataframe to visualize the results
    eval_df = pd.DataFrame([[accuracy, f1, precision, recall]], columns=['accuracy',
                                                'f1_score', 'precision', 'recall'])
    return eval_df
```

In [49]:
```python
lg = LogisticRegression(max_iter=40)

results = train_evaluate_model(lg, X_train, y_train, X_test, y_test)

results.index = ['LogisticRegression']
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

In [50]:
```python
decision_tree = DecisionTreeClassifier(max_leaf_nodes=10)
decision_tree_results = train_evaluate_model(decision_tree,X_train, y_train, X_test, y_test)

decision_tree_results.index = ['DecisionTree']

results = results.append(decision_tree_results)
```

In [51]:
```python
KNN = KNeighborsClassifier(n_neighbors=11)
```

In [51]:
```
KNN = KNeighborsClassifier(n_neighbors=11)

knn = train_evaluate_model(KNN, X_train, y_train, X_test, y_test)
knn.index =['KNearsNeighbors']
results = results.append(knn)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning:

Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts
along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the
statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid
this warning.
```

In [52]:
```
rfc = RandomForestClassifier(n_estimators=10)

rfc_result = train_evaluate_model(rfc, X_train, y_train, X_test, y_test)
rfc_result.index = ['RandomForest']

results = results.append(rfc_result)
```

In [53]:
```
Naive_Bayes = GaussianNB()
Naive_Bayes_result = train_evaluate_model(Naive_Bayes, X_train, y_train, X_test, y_test)
Naive_Bayes_result.index = ['NaiveBayes']

results = results.append(Naive_Bayes_result)
```

5. Evaluate

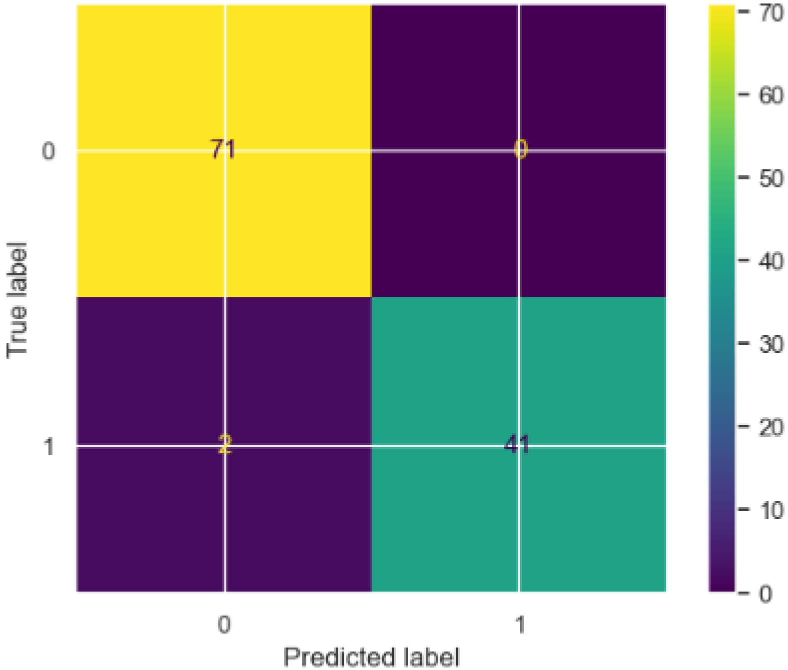
In [54]:
```
results.head(6).sort_values(by='f1_score',ascending=False).style.background_gradient(cmap = sns.color_palette("ch:s=-.2,r=.6", a
```

Out[54]:

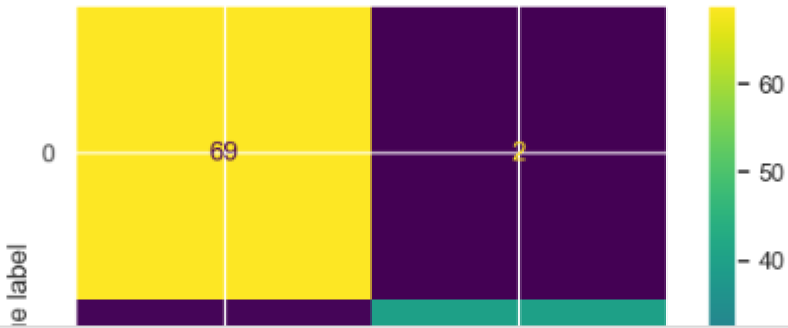
	accuracy	f1_score	precision	recall
KNearsNeighbors	0.982456	0.976190	1.000000	0.953488
RandomForest	0.973684	0.963855	1.000000	0.930233
NaiveBayes	0.973684	0.963855	1.000000	0.930233
LogisticRegression	0.964912	0.952381	0.975610	0.930233
DecisionTree	0.956140	0.941176	0.952381	0.930233

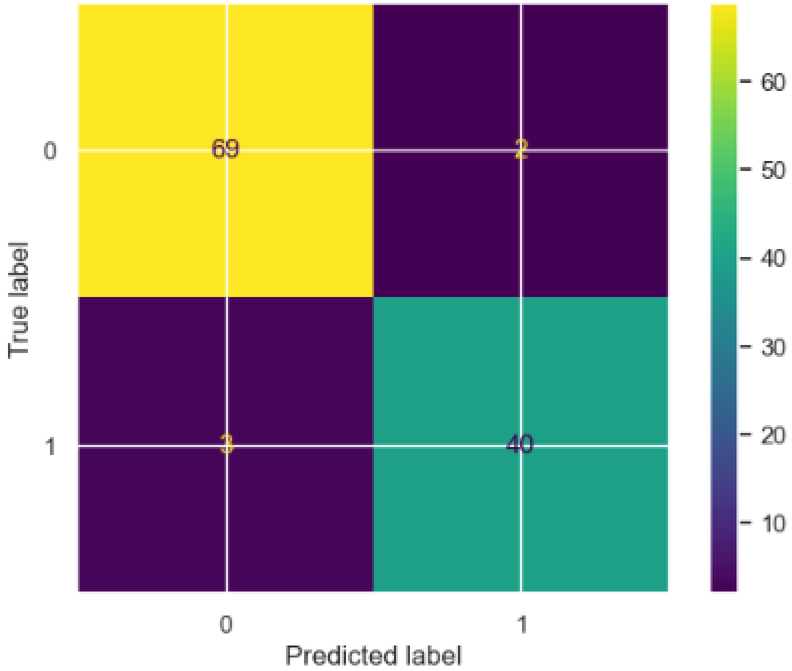
In [55]:
```
ConfusionMatrixDisplay.from_estimator(KNN , X_test , y_test);
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning:

Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts
along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the
statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid
this warning.
```



In [56]:
```
ConfusionMatrixDisplay.from_estimator(decision_tree , X_test , y_test);
```





6. Communicate

Now that we have a reasonable model, let's graph the importance of each feature.

A horizontal bar chart with the 10 most important features for `DecisionTree model`

A horizontal bar chart with the 10 most important features for `DecisionTree model`

In [57]:
```
# Get importances
importances = decision_tree.feature_importances_

# Put importances into a Series
feat_imp = pd.Series(importances , index=X_train.columns).sort_values()

# Plot series
feat_imp.tail(10).plot(kind='barh')
plt.xlabel("Gini Importance")
plt.ylabel("Feature")
plt.title("model_over Feature Importance");
```