```
pip install sdv
```

```
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch>=1.11.0->ctgan>=0.10.0->sdv)
  Using cached nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-curand-cu12==10.3.2.106 (from torch>=1.11.0->ctgan>=0.10.0->sdv)
  Using cached nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch>=1.11.0->ctgan>=0.10.0->sdv)
  Using cached nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cusparse-cu12==12.1.0.106 (from torch>=1.11.0->ctgan>=0.10.0->sdv)
  Using cached nvidia_cusparse_cu12-12.1.0.106-py3-none-manylinux1_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-nccl-cu12==2.20.5 (from torch>=1.11.0->ctgan>=0.10.0->sdv)
  Using cached nvidia_nccl_cu12-2.20.5-py3-none-manylinux2014_x86_64.whl.metadata (1.8 kB)
Collecting nvidia-nvtx-cu12==12.1.105 (from torch>=1.11.0->ctgan>=0.10.0->sdv)
  Using cached nvidia_nvtx_cu12-12.1.105-py3-none-manylinux1_x86_64.whl.metadata (1.7 kB)
Requirement already satisfied: triton==2.3.1 in /usr/local/lib/python3.10/dist-packages (from torch>=1.11.0->ctgan>=0.10.0->sdv)
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-cu12==11.4.5.107->torch>=1.11.0->ctgan>=0.10.0->sdv)
  Downloading nvidia_nvjitlink_cu12-12.5.82-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.11.0->ctgan>=0.
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy->torch>=1.11.0->ctgan>=
Downloading sdv-1.15.0-py3-none-any.whl (146 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 146.4/146.4 kB 8.5 MB/s eta 0:00:00
Downloading boto3-1.34.149-py3-none-any.whl (139 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 139.2/139.2 kB 8.4 MB/s eta 0:00:00
Downloading botocore-1.34.149-py3-none-any.whl (12.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.4/12.4 MB 63.1 MB/s eta 0:00:00
Downloading copulas-0.11.0-py3-none-any.whl (51 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 51.9/51.9 kB 3.3 MB/s eta 0:00:00
Downloading ctgan-0.10.1-py3-none-any.whl (24 kB)
Downloading deepecho-0.6.0-py3-none-any.whl (27 kB)
Downloading rdt-1.12.2-py3-none-any.whl (65 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 65.2/65.2 kB 4.0 MB/s eta 0:00:00
Downloading sdmetrics-0.15.0-py3-none-any.whl (170 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 170.5/170.5 kB 10.6 MB/s eta 0:00:00
Downloading Faker-26.0.0-py3-none-any.whl (1.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 48.9 MB/s eta 0:00:00
Downloading jmespath-1.0.1-py3-none-any.whl (20 kB)
Downloading plotly-5.23.0-py3-none-any.whl (17.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.3/17.3 MB 50.4 MB/s eta 0:00:00
Downloading s3transfer-0.10.2-py3-none-any.whl (82 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 82.7/82.7 kB 5.7 MB/s eta 0:00:00
Using cached nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl (410.6 MB)
Using cached nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (14.1 MB)
Using cached nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (23.7 MB)
Using cached nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (823 kB)
Using cached nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_64.whl (731.7 MB)
Using cached nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl (121.6 MB)
Using cached nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x86_64.whl (56.5 MB)
Using cached nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1_x86_64.whl (124.2 MB)
Using cached nvidia_cusparse_cu12-12.1.0.106-py3-none-manylinux1_x86_64.whl (196.0 MB)
Using cached nvidia_nccl_cu12-2.20.5-py3-none-manylinux2014_x86_64.whl (176.2 MB)
Using cached nvidia_nvtx_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (99 kB)
Downloading nvidia_nvjitlink_cu12-12.5.82-py3-none-manylinux2014_x86_64.whl (21.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.3/21.3 MB 12.2 MB/s eta 0:00:00
Installing collected packages: plotly, nvidia-nvtx-cu12, nvidia-nvjitlink-cu12, nvidia-nccl-cu12, nvidia-curand-cu12, nvidia-cuf
  Attempting uninstall: plotly
    Found existing installation: plotly 5.15.0
    Uninstalling plotly-5.15.0:
      Successfully uninstalled plotly-5.15.0
Successfully installed Faker-26.0.0 boto3-1.34.149 botocore-1.34.149 copulas-0.11.0 ctgan-0.10.1 deepecho-0.6.0 jmespath-1.0.1 n
```

```
pip install table_evaluator
```

```
Collecting table_evaluator
  Downloading table_evaluator-1.6.1-py3-none-any.whl.metadata (8.8 kB)
Requirement already satisfied: pandas==2.0.* in /usr/local/lib/python3.10/dist-packages (from table_evaluator) (2.0.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from table_evaluator) (1.25.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from table_evaluator) (4.66.4)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from table_evaluator) (5.9.5)
Collecting dython==0.7.3 (from table_evaluator)
  Downloading dython-0.7.3-py3-none-any.whl.metadata (3.0 kB)
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (from table_evaluator) (0.13.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from table_evaluator) (3.7.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from table_evaluator) (1.3.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from table_evaluator) (1.13.1)
Collecting scikit-plot>=0.3.7 (from dython==0.7.3->table_evaluator)
  Downloading scikit_plot-0.3.7-py3-none-any.whl.metadata (7.1 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas==2.0.*->table_evaluat
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas==2.0.*->table_evaluator) (2024.
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas==2.0.*->table_evaluator) (202
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->table_evaluator) (1.2.
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->table_evaluator) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->table_evaluator) (4.5
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->table_evaluator) (1.4
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->table_evaluator) (24.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->table_evaluator) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->table_evaluator) (3.1.
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->table_evaluator) (1.4.2
```

```
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->table_evaluator)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas==2.0.*->tab
Downloading table_evaluator-1.6.1-py3-none-any.whl (22 kB)
Downloading dython-0.7.3-py3-none-any.whl (23 kB)
Downloading scikit_plot-0.3.7-py3-none-any.whl (33 kB)
Installing collected packages: scikit-plot, dython, table_evaluator
Successfully installed dython-0.7.3 scikit-plot-0.3.7 table_evaluator-1.6.1
```

```
import pandas as pd

from sdmetrics.reports.single_table import QualityReport
from ctgan import CTGAN

from rdt import HyperTransformer
```

```
real_data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/CSV_FILE/breast_cancer.csv")
```

```
df = pd.DataFrame(real_data)

print(df.columns)

print("Original DataFrame:")
print(df)

# Dropping column 'B'
df = df.drop(['id','Unnamed: 32'], axis=1)

print("\nDataFrame after dropping column 'Id':")
print(df)
```

```
567          0.2650          0.4087           0.12400
568          0.0000          0.2871           0.07039

[569 rows x 31 columns]
```

```
NUM_ROWS = 100000
NUM_EPOCHS = 1000
BATCH_SIZE = 100
```

```
df.shape
```

```
(569, 31)
```

```
ht = HyperTransformer()
ht.detect_initial_config(data = df)
detected_config = ht.get_config()
display(detected_config)
```


```
{
    "sdtypes": {
        "diagnosis": "categorical",
        "radius_mean": "numerical",
        "texture_mean": "numerical",
        "perimeter_mean": "numerical",
        "area_mean": "numerical",
        "smoothness_mean": "numerical",
        "compactness_mean": "numerical",
        "concavity_mean": "numerical",
        "concave points_mean": "numerical",
        "symmetry_mean": "numerical",
        "fractal_dimension_mean": "numerical",
        "radius_se": "numerical",
        "texture_se": "numerical",
        "perimeter_se": "numerical",
        "area_se": "numerical",
        "smoothness_se": "numerical",
        "compactness_se": "numerical",
        "concavity_se": "numerical",
        "concave points_se": "numerical",
        "symmetry_se": "numerical",
        "fractal_dimension_se": "numerical",
        "radius_worst": "numerical",
        "texture_worst": "numerical",
        "perimeter_worst": "numerical",
        "area_worst": "numerical",
        "smoothness_worst": "numerical",
        "compactness_worst": "numerical",
        "concavity_worst": "numerical",
        "concave points_worst": "numerical",
        "symmetry_worst": "numerical",
        "fractal_dimension_worst": "numerical"
    },
    "transformers": {
        "diagnosis": UniformEncoder(),
        "radius_mean": FloatFormatter(),
        "texture_mean": FloatFormatter(),
        "perimeter_mean": FloatFormatter(),
        "area_mean": FloatFormatter(),
        "smoothness_mean": FloatFormatter(),
        "compactness_mean": FloatFormatter(),
        "concavity_mean": FloatFormatter(),
        "concave points_mean": FloatFormatter(),
        "symmetry_mean": FloatFormatter(),
        "fractal_dimension_mean": FloatFormatter(),
        "radius_se": FloatFormatter(),
        "texture_se": FloatFormatter(),
        "perimeter_se": FloatFormatter(),
        "area_se": FloatFormatter(),
        "smoothness_se": FloatFormatter(),
        "compactness_se": FloatFormatter(),
        "concavity_se": FloatFormatter(),
        "concave points_se": FloatFormatter(),
        "symmetry_se": FloatFormatter(),
        "fractal_dimension_se": FloatFormatter(),
        "radius_worst": FloatFormatter(),
        "texture_worst": FloatFormatter(),
        "perimeter_worst": FloatFormatter(),
        "area_worst": FloatFormatter(),
        "smoothness_worst": FloatFormatter(),
        "compactness_worst": FloatFormatter(),
        "concavity_worst": FloatFormatter(),
        "concave points_worst": FloatFormatter(),
        "symmetry_worst": FloatFormatter(),
        "fractal_dimension_worst": FloatFormatter()
    }
}
```

```
ht.fit(df)
transformed_df = ht.transform(df)
transformed_df
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean
0	0.132427	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710
1	0.119307	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017
2	0.344664	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790
3	0.013330	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520
4	0.130511	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430
...	...	...	...	...	...	...	...	...	...
564	0.222094	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890
565	0.004383	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791
566	0.249081	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302
567	0.228957	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200
568	0.719430	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000

569 rows × 31 columns

```
import time

start_time = time.time()  # Capture start time before training

model = CTGAN(
    epochs=NUM_EPOCHS,
    verbose=True,
    batch_size=BATCH_SIZE,
    embedding_dim = 1024,
    discriminator_steps = 6,
    discriminator_dim = (512,512)
)

model.fit(transformed_df)

# Training is finished, record end time
end_time = time.time()

# Calculate total training time in seconds
training_time = end_time - start_time

print(f"Training completed! Total time taken: {training_time:.2f} seconds")

model.save("/content/drive/MyDrive/Colab Notebooks/CSV_FILE/Models/breast_cancer_1000epochs_100BS_1024_6_512.pkl")
```

```
Gen. (-1.01) | Discrim. (-1.29): 100%|██████████| 1000/1000 [42:23<00:00,  2.54s/it]Training completed! Total time taken: 2559.97 s
```

```
from sdv.metadata import SingleTableMetadata
metadata = SingleTableMetadata()
metadata.detect_from_dataframe(df)
metadata_dict= metadata.to_dict()
metadata.visualize()
```

```
diagnosis : categorical
radius_mean : numerical
texture_mean : numerical
perimeter_mean : numerical
area_mean : numerical
smoothness_mean : numerical
compactness_mean : numerical
concavity_mean : numerical
concave points_mean : numerical
symmetry_mean : numerical
fractal_dimension_mean : numerical
radius_se : numerical
texture_se : numerical
perimeter_se : numerical
area_se : numerical
smoothness_se : numerical
compactness_se : numerical
concavity_se : numerical
concave points_se : numerical
symmetry_se : numerical
fractal_dimension_se : numerical
radius_worst : numerical
texture_worst : numerical
perimeter_worst : numerical
area_worst : numerical
smoothness_worst : numerical
compactness_worst : numerical
concavity_worst : numerical
concave points_worst : numerical
symmetry_worst : numerical
fractal_dimension_worst : numerical
```

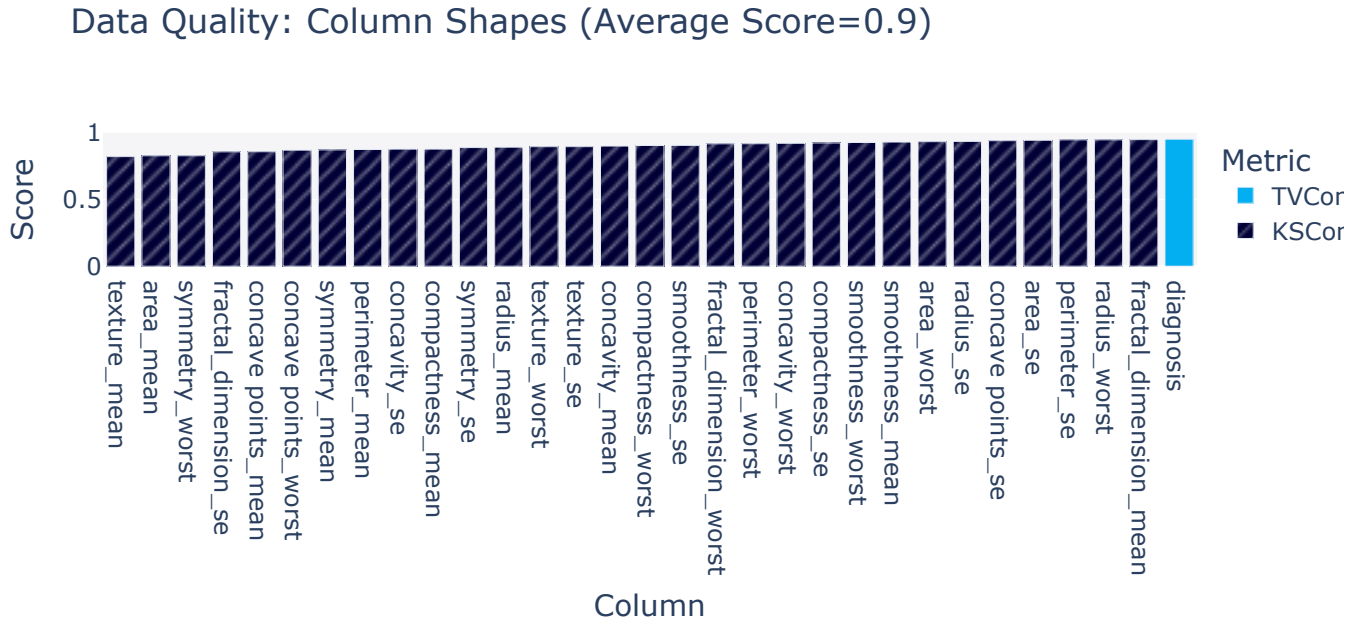
```
categorical_columns = [column for column, info in metadata_dict['columns'].items() if info['sdtype'] == 'categorical']
print(categorical_columns)
```

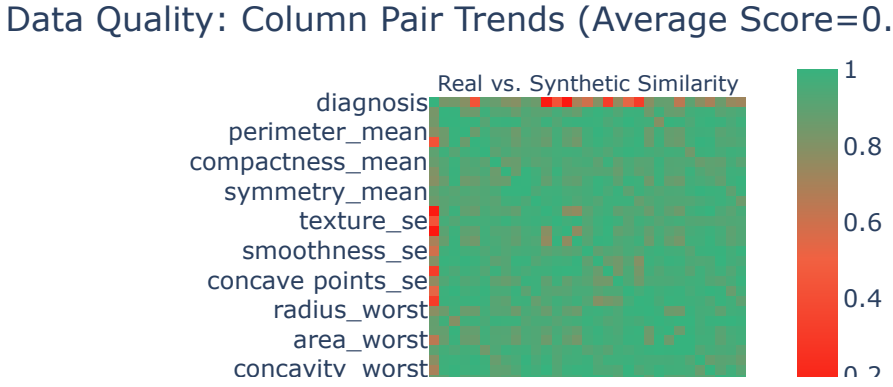
```
['diagnosis']
```

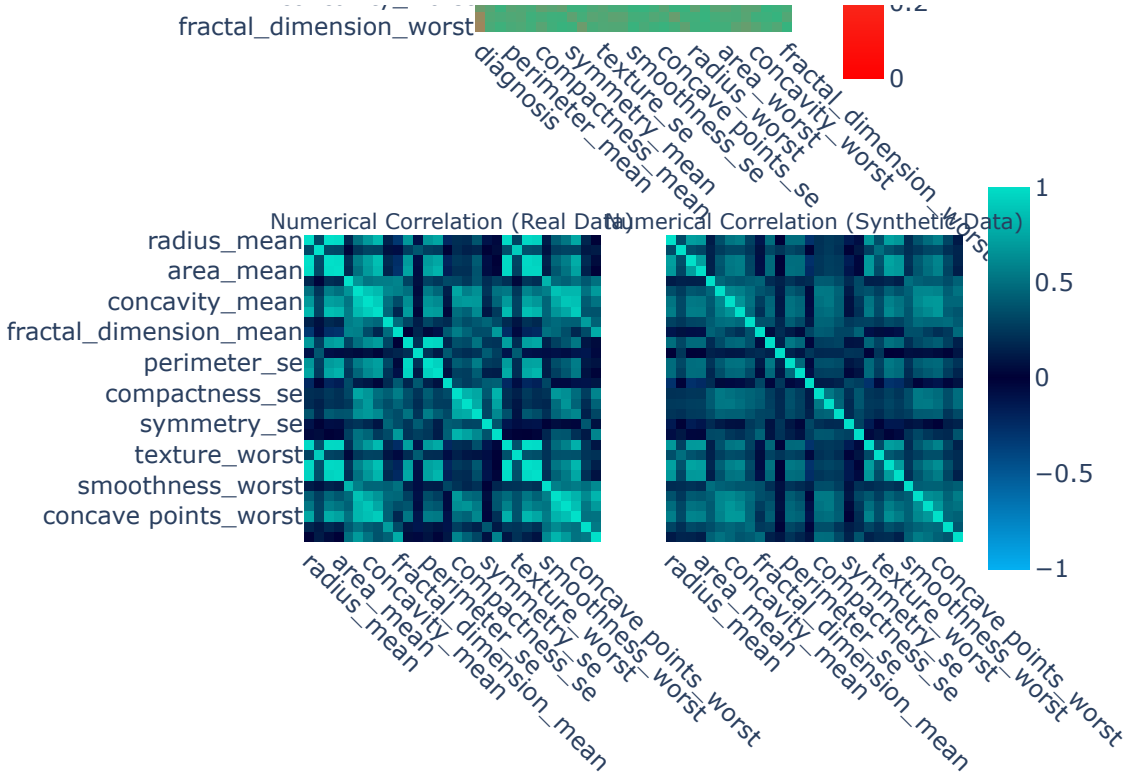
```
from sdmetrics.reports.single_table import QualityReport

# Get Synthetic data
synthetic_data = model.sample(NUM_ROWS)
# reverse transform the data
synthetic_data = ht.reverse_transform(synthetic_data)

report = QualityReport()
# Use the metadata OBJECT instead of the dictionary
report.generate(df, synthetic_data, metadata.to_dict())

cs_report = report.get_details(property_name="Column Shapes")
print(cs_report)

# Create the first figure
fig1 = report.get_visualization(property_name='Column Shapes')
fig1.show()

# Create the second figure
fig2 = report.get_visualization(property_name='Column Pair Trends')

fig2.show()

report.save(filepath='/content/drive/MyDrive/Colab Notebooks/CSV_FILE/Models/breast_cancer_report_1000epochs_100BS_1024_6_512.pkl')
```

```
Generating report ...

(1/2) Evaluating Column Shapes: |██████████| 31/31 [00:00<00:00, 57.15it/s]|
Column Shapes Score: 90.14%

(2/2) Evaluating Column Pair Trends: |██████████| 465/465 [00:19<00:00, 23.29it/s]|
Column Pair Trends Score: 92.36%

Overall Score (Average): 91.25%
```

	Column	Metric	Score
0	diagnosis	TVComplement	0.952057
1	radius_mean	KSComplement	0.892207
2	texture_mean	KSComplement	0.821577
3	perimeter_mean	KSComplement	0.875094
4	area_mean	KSComplement	0.828137
5	smoothness_mean	KSComplement	0.930931
6	compactness_mean	KSComplement	0.878031
7	concavity_mean	KSComplement	0.900935
8	concave points_mean	KSComplement	0.858422
9	symmetry_mean	KSComplement	0.872187
10	fractal_dimension_mean	KSComplement	0.949024
11	radius_se	KSComplement	0.935554
12	texture_se	KSComplement	0.896307
13	perimeter_se	KSComplement	0.947641
14	area_se	KSComplement	0.943967
15	smoothness_se	KSComplement	0.905786
16	compactness_se	KSComplement	0.924217
17	concavity_se	KSComplement	0.877905
18	concave points_se	KSComplement	0.941071
19	symmetry_se	KSComplement	0.889423
20	fractal_dimension_se	KSComplement	0.857790
21	radius_worst	KSComplement	0.947704
22	texture_worst	KSComplement	0.896099
23	perimeter_worst	KSComplement	0.918379
24	area_worst	KSComplement	0.933697
25	smoothness_worst	KSComplement	0.927583
26	compactness_worst	KSComplement	0.905088
27	concavity_worst	KSComplement	0.921396
28	concave points_worst	KSComplement	0.870047
29	symmetry_worst	KSComplement	0.828184
30	fractal_dimension_worst	KSComplement	0.917936



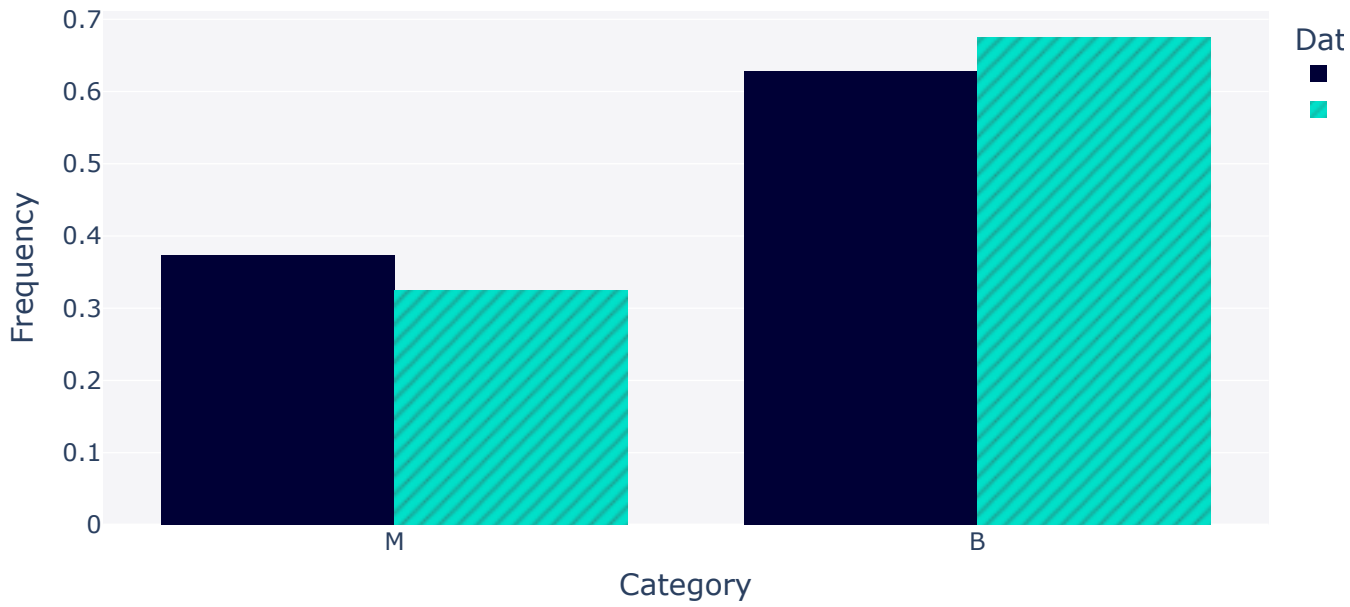


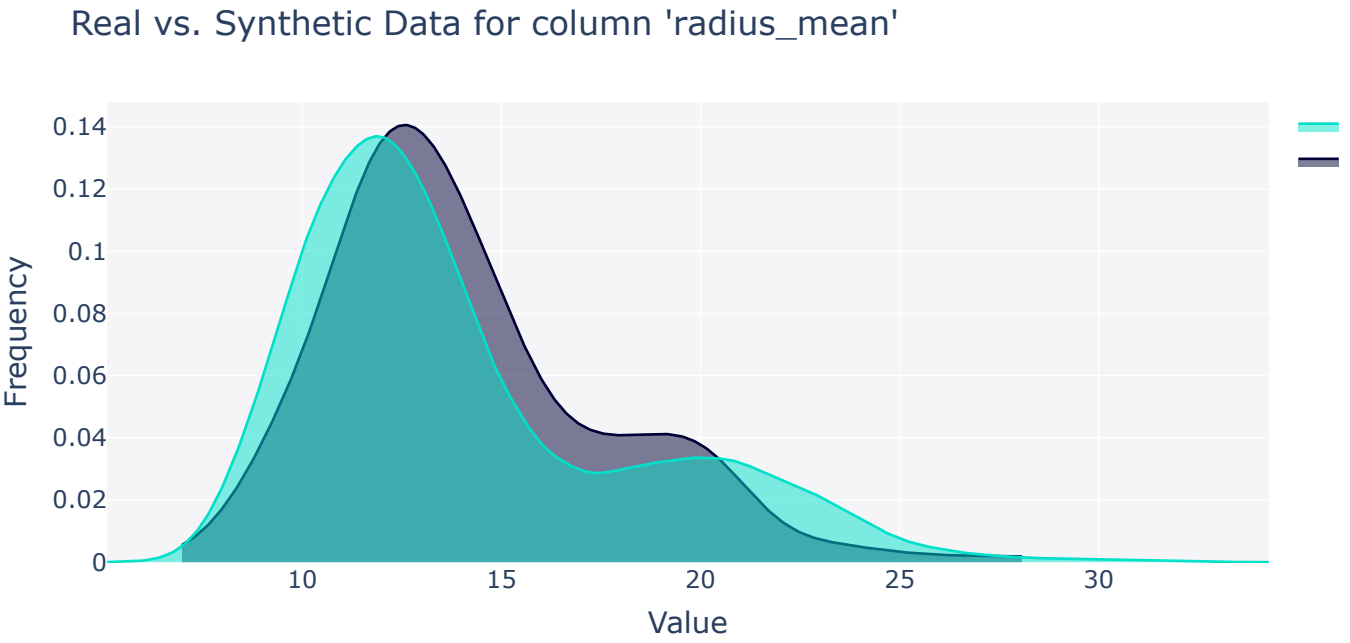


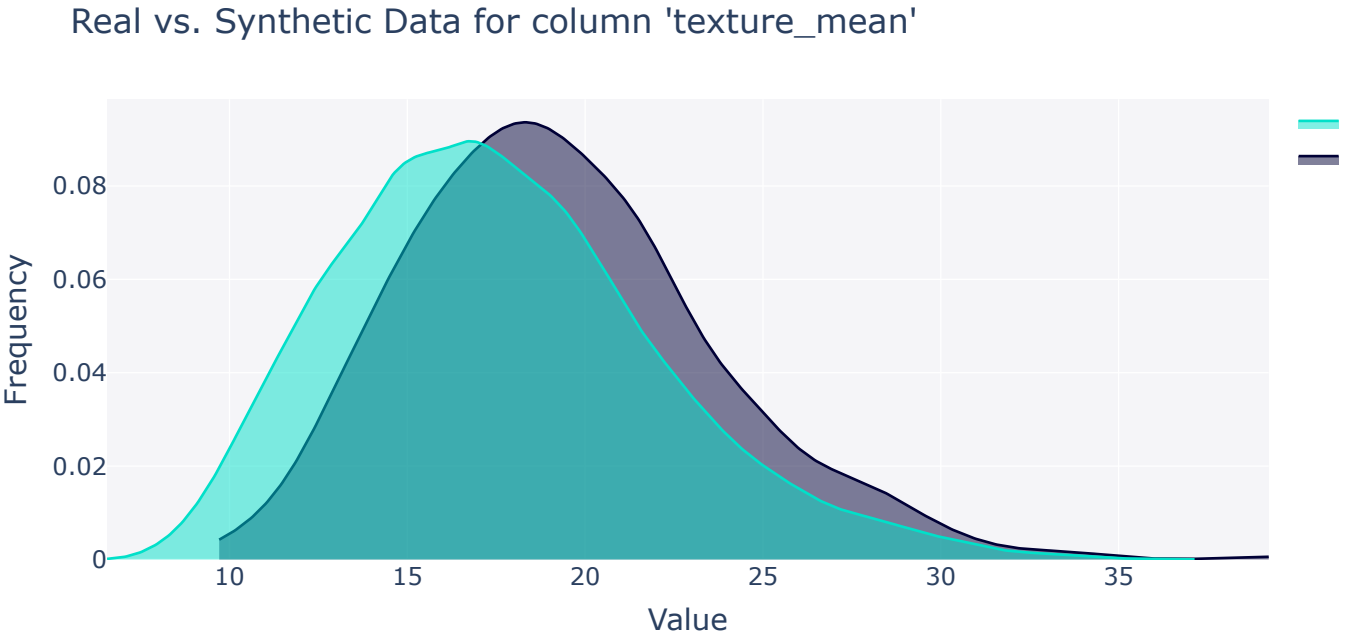

```
from sdmetrics.single_column import CSTest

for column in categorical_columns:
    cstest_result = CSTest.compute(
        real_data=df[column],
        synthetic_data=synthetic_data[column]
    )
    print(f"CSTest for column {column}: {cstest_result}")
```

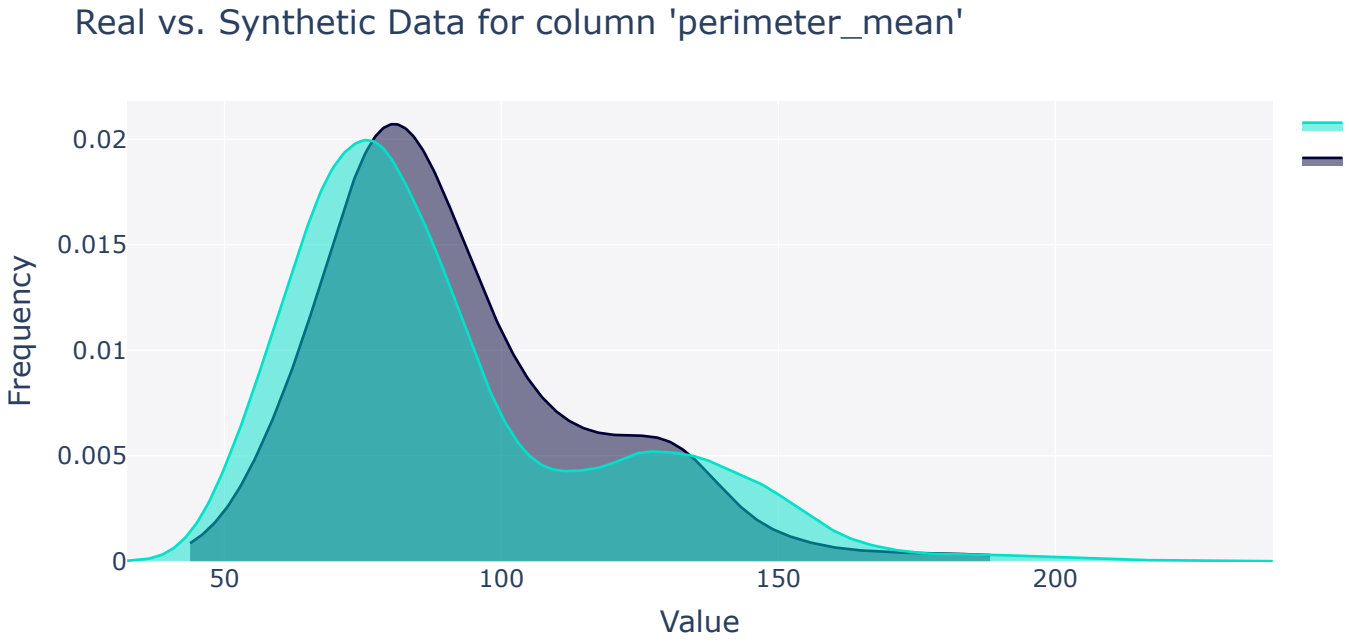
```
CSTest for column diagnosis: 0.9210106422771743
```

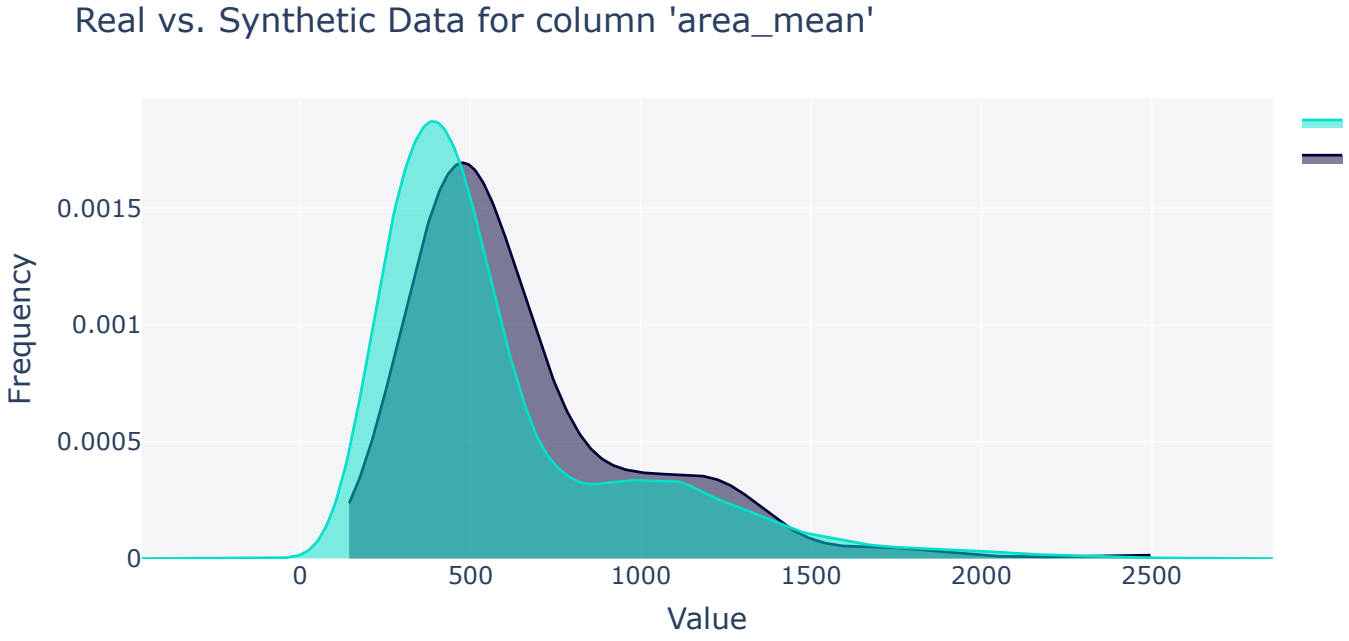
```
from sdmetrics.visualization import get_column_plot

# Loop through each column in the dataframe
for column in df.columns:
    fig = get_column_plot(
        real_data=df,
        synthetic_data=synthetic_data,
        column_name=column,
    )

    fig.show()
```

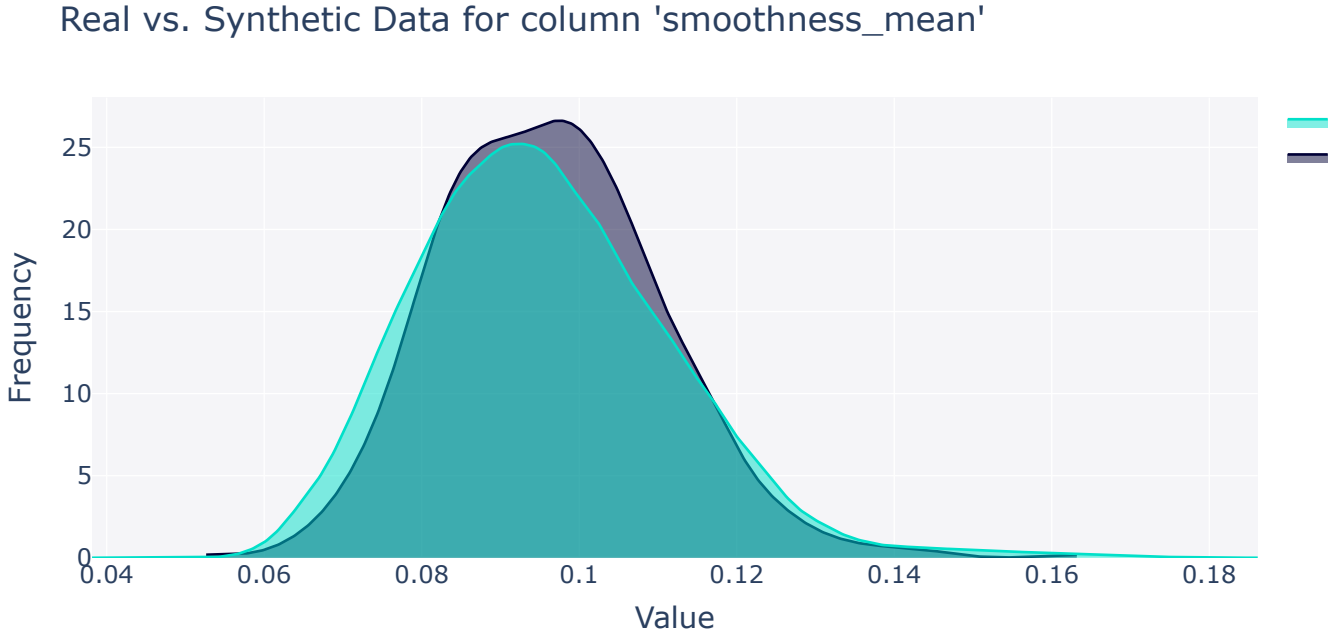

Real vs. Synthetic Data for column 'diagnosis'
Frequency
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
M
B
Category
Dat

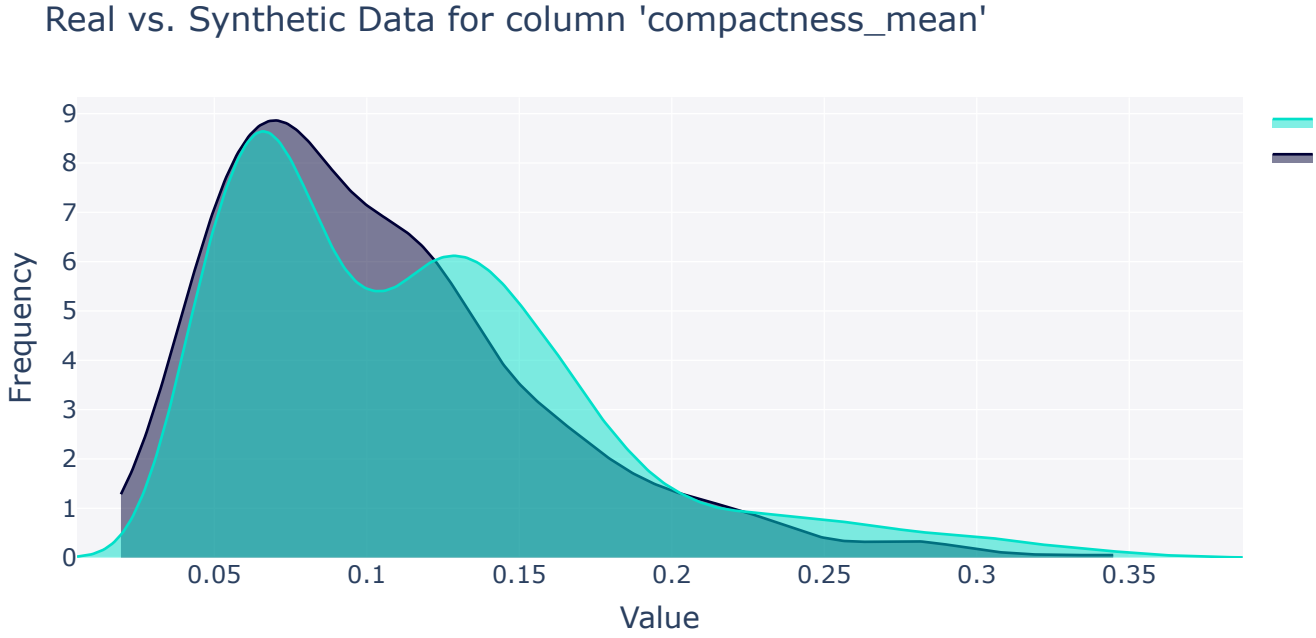

Real vs. Synthetic Data for column 'radius_mean'
Frequency
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
10
15
20
25
30
Value

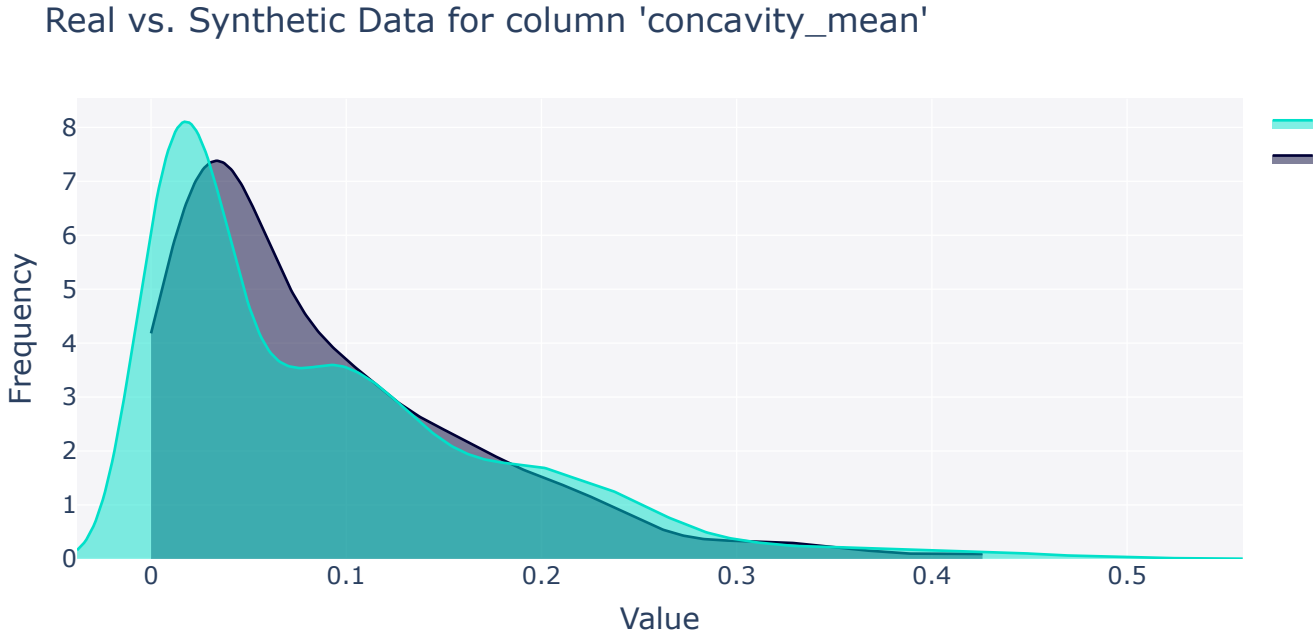

Real vs. Synthetic Data for column 'texture_mean'
Frequency
0.08
0.06
0.04
0.02
0
10
15
20
25
30
35
Value

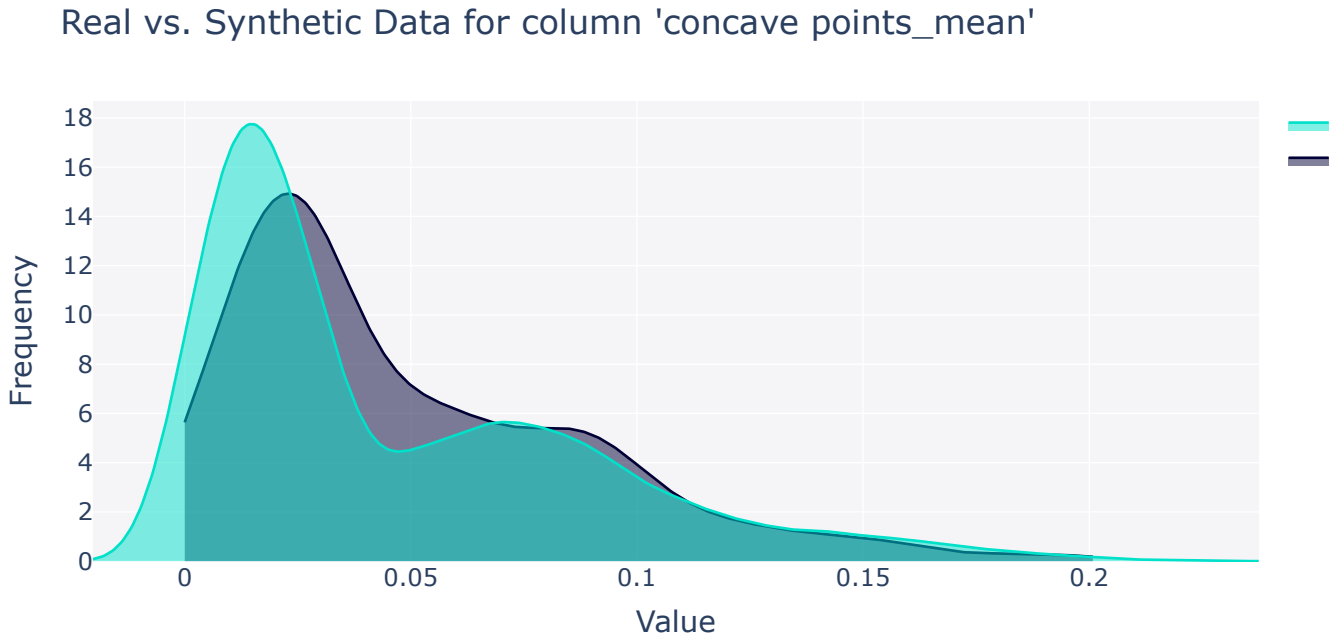

Real vs. Synthetic Data for column 'perimeter_mean'
Frequency
0.02
0.015
0.01
0.005
0
50
100
150
200
Value

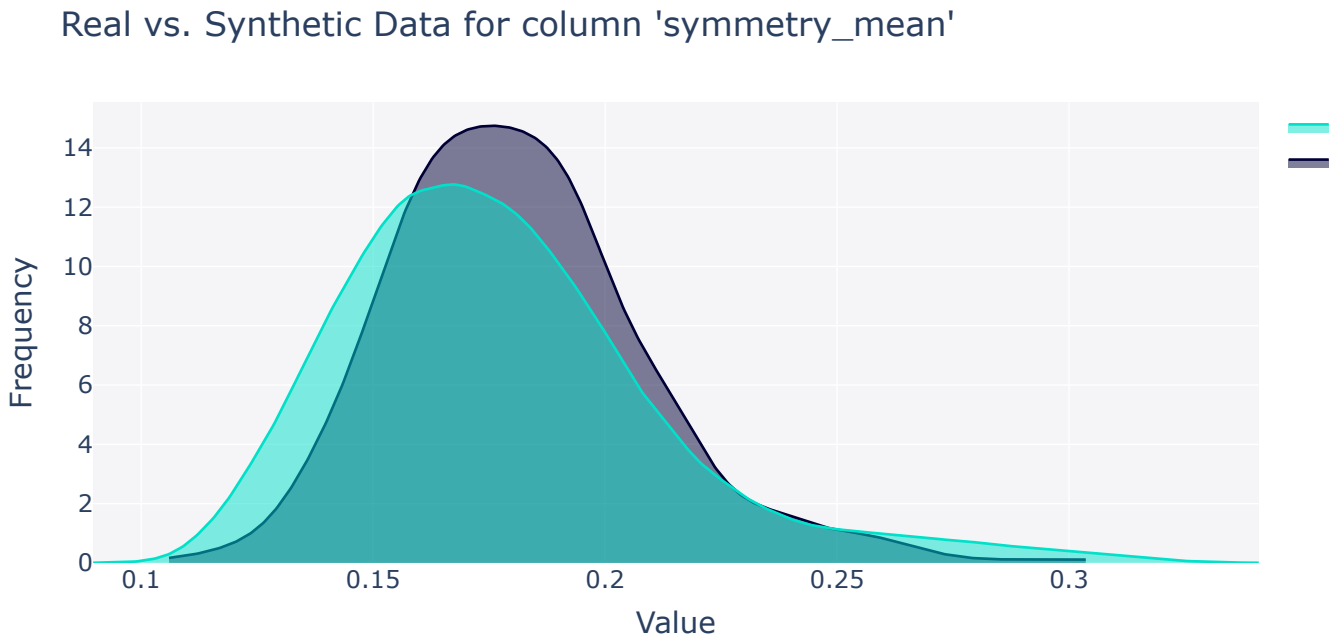

Real vs. Synthetic Data for column 'area_mean'
Frequency
0.0015
0.001
0.0005
0
0
500
1000
1500
2000
2500
Value

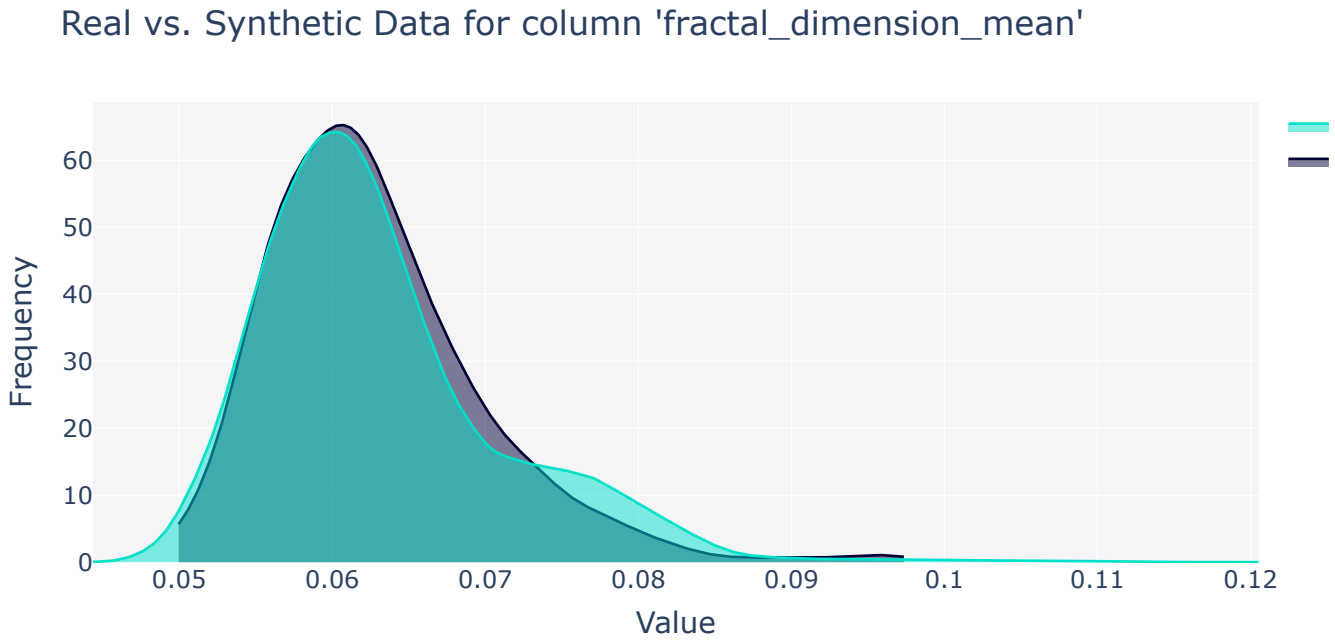

Real vs. Synthetic Data for column 'smoothness_mean'
Frequency
25
20
15
10
5
0
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
Value

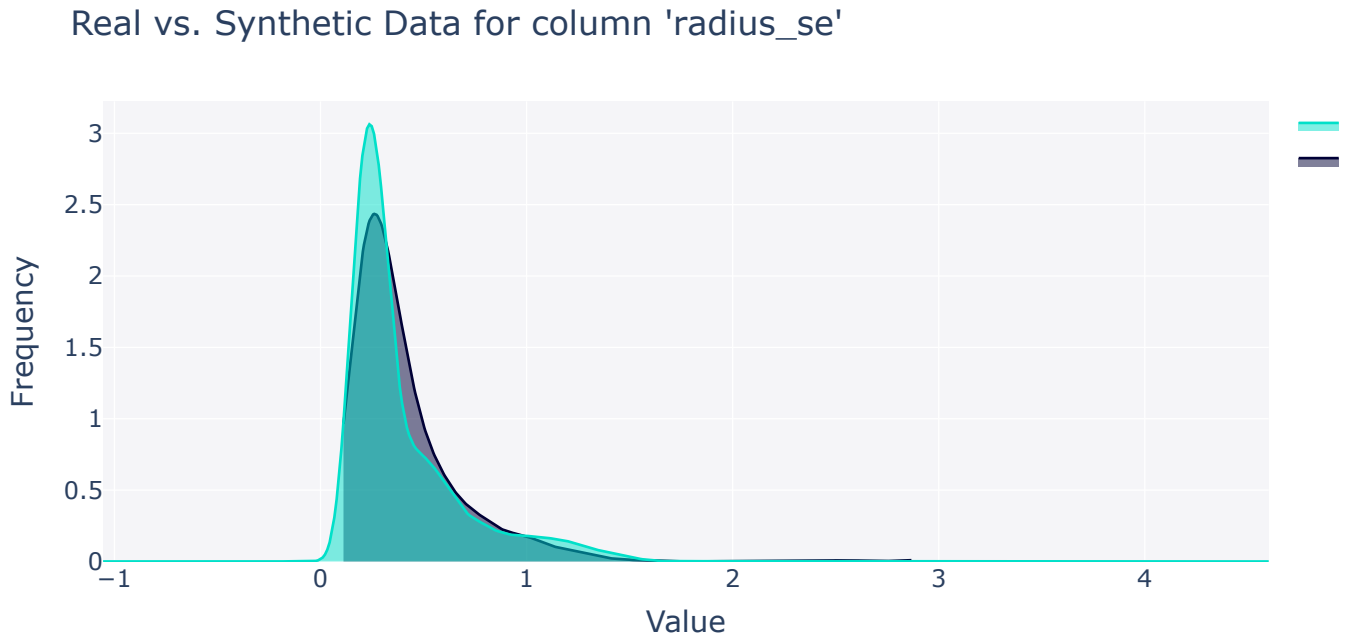

Real vs. Synthetic Data for column 'compactness_mean'
Frequency
Value
0
1
2
3
4
5
6
7
8
9
0.05
0.1
0.15
0.2
0.25
0.3
0.35

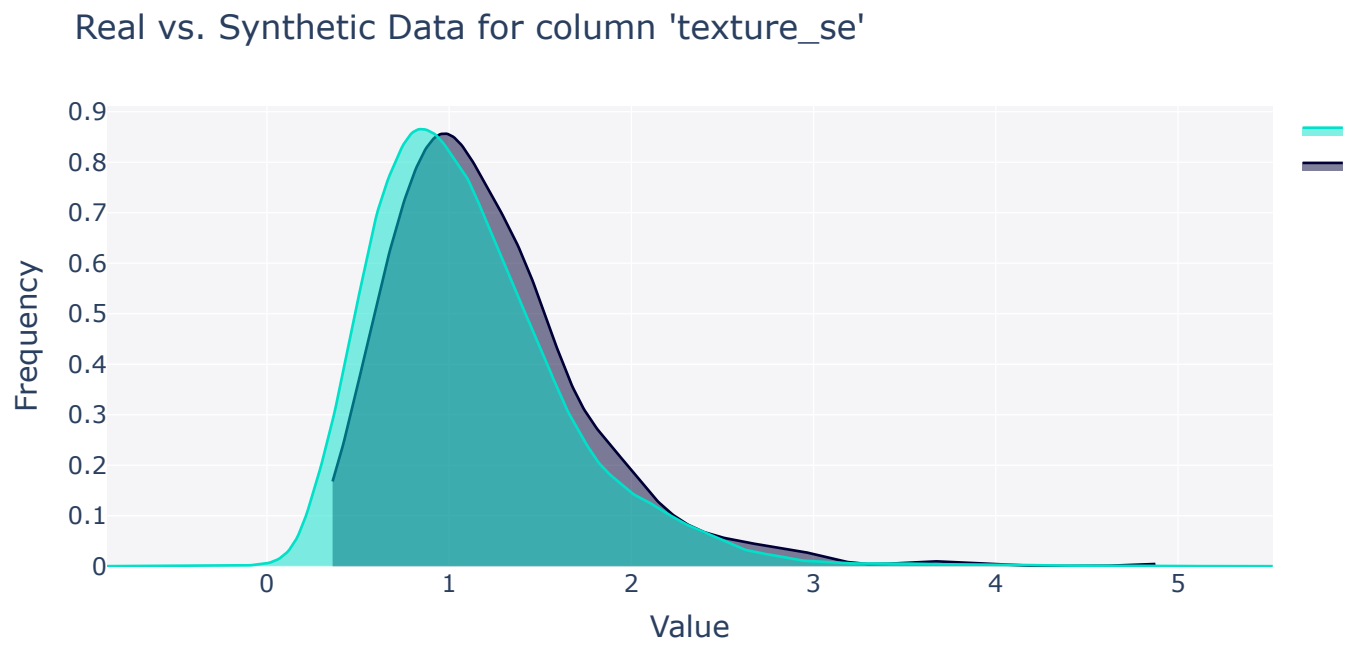

Real vs. Synthetic Data for column 'concavity_mean'
Frequency
Value
0
1
2
3
4
5
6
7
8
0
0.1
0.2
0.3
0.4
0.5

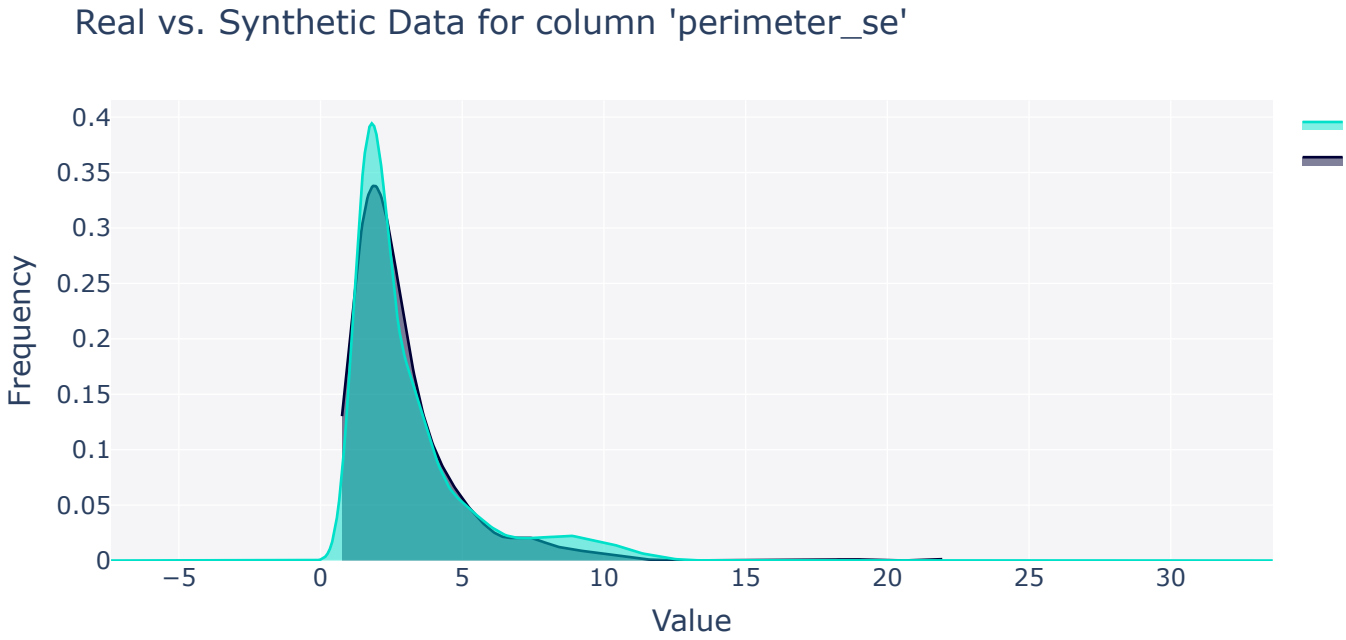

Real vs. Synthetic Data for column 'concave points_mean'
Frequency
Value
0
2
4
6
8
10
12
14
16
18
0
0.05
0.1
0.15
0.2

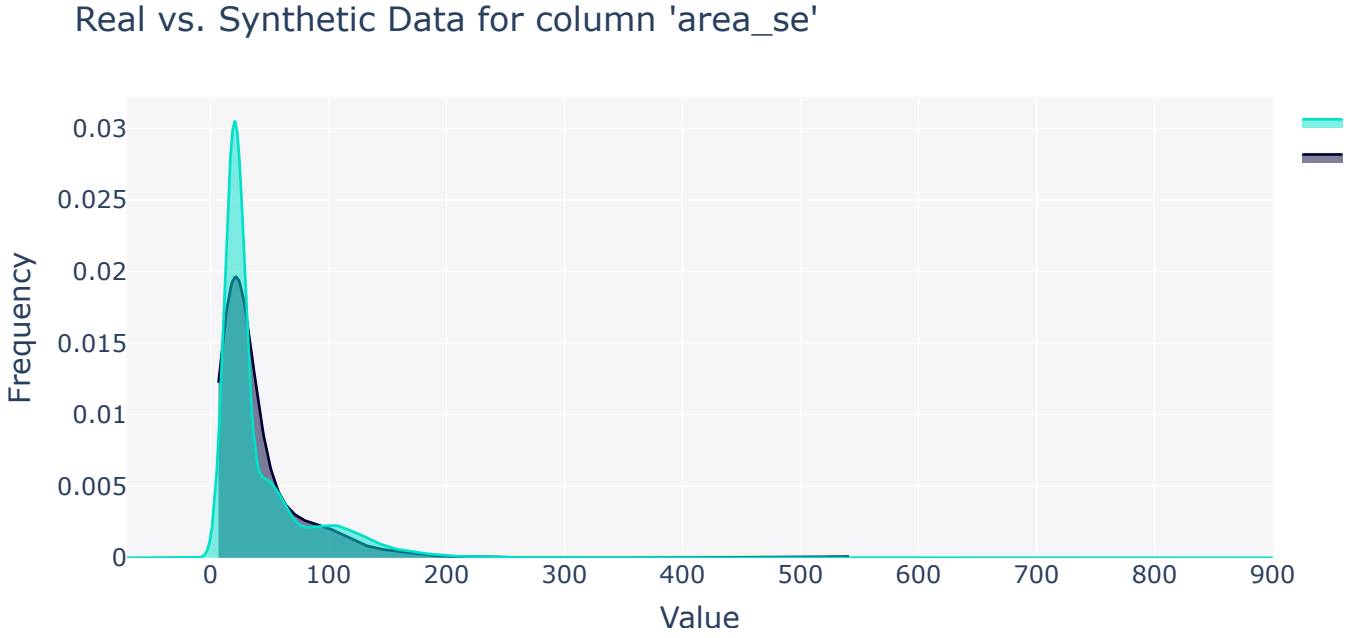

Real vs. Synthetic Data for column 'symmetry_mean'
Frequency
0
2
4
6
8
10
12
14
0.1
0.15
0.2
0.25
0.3
Value

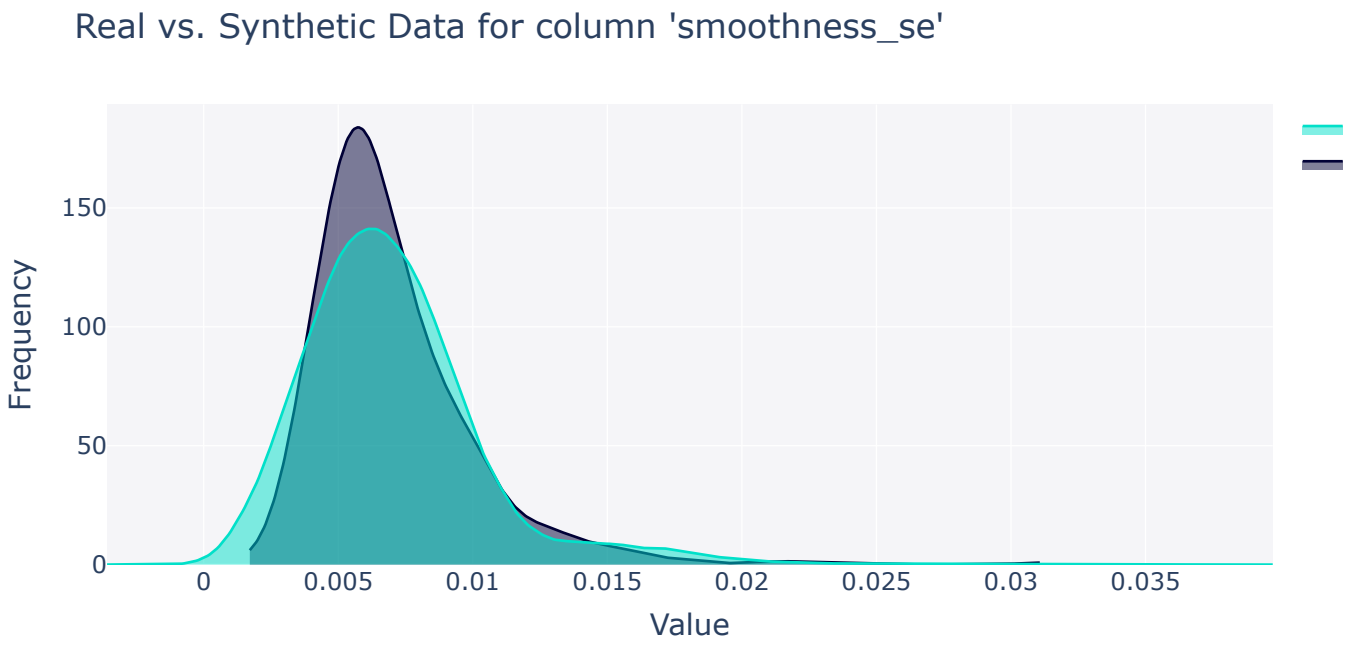

Real vs. Synthetic Data for column 'fractal_dimension_mean'
Frequency
0
10
20
30
40
50
60
0.05
0.06
0.07
0.08
0.09
0.1
0.11
0.12
Value

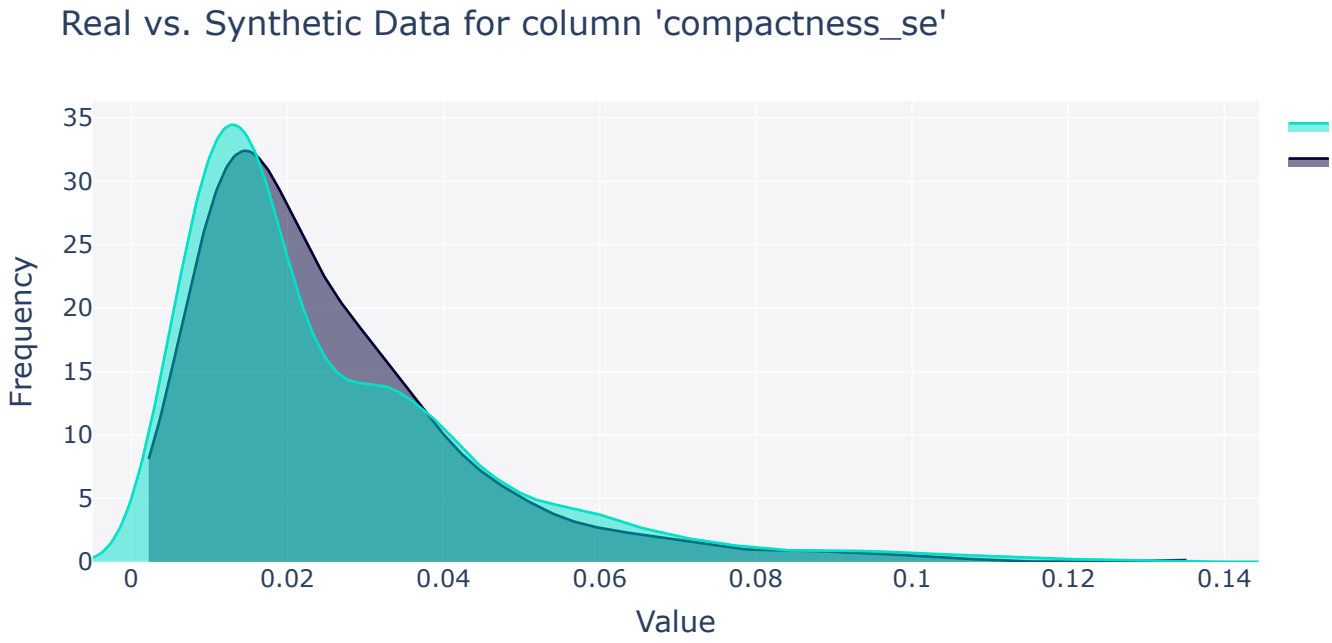

Real vs. Synthetic Data for column 'radius_se'
Frequency
0
0.5
1
1.5
2
2.5
3
−1
0
1
2
3
4
Value

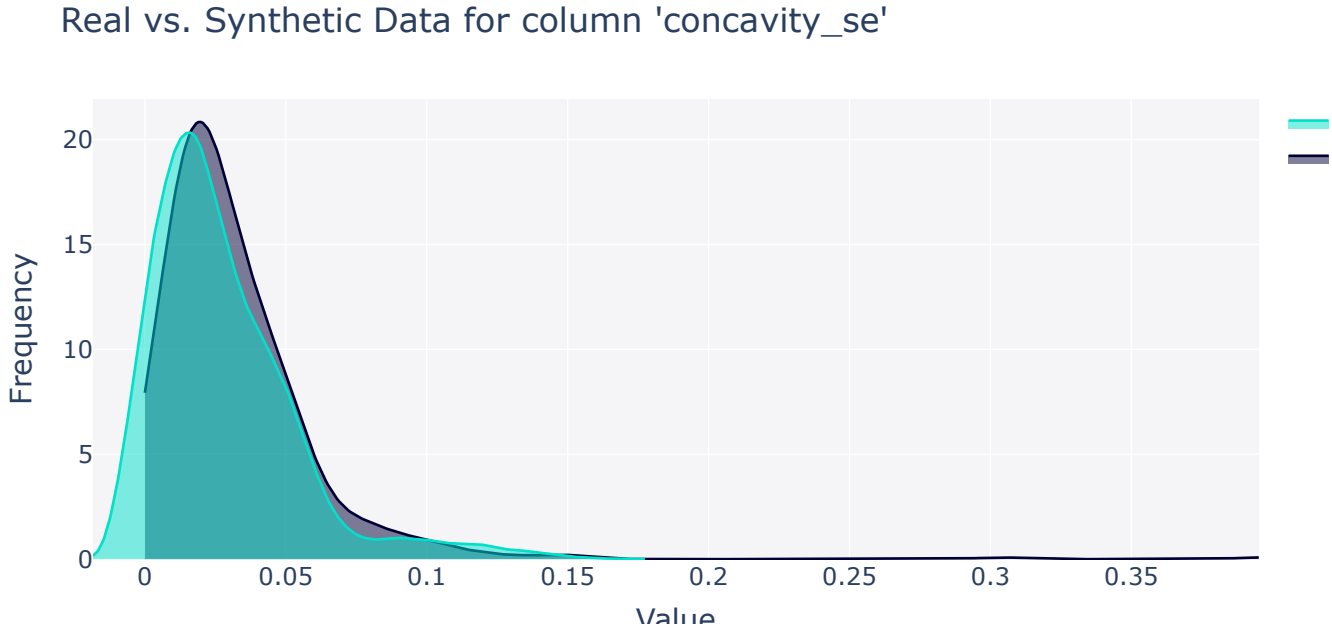

Real vs. Synthetic Data for column 'texture_se'
Frequency
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
1
2
3
4
5
Value

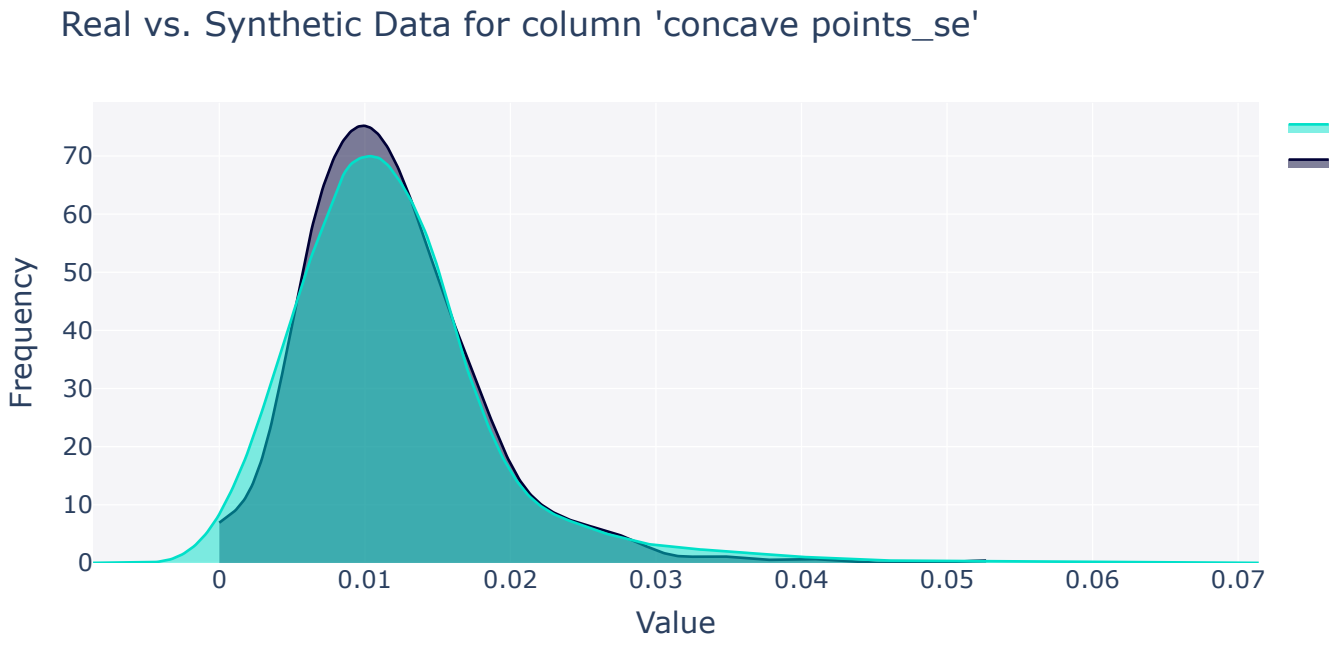

Real vs. Synthetic Data for column 'perimeter_se'
Frequency
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
−5
0
5
10
15
20
25
30
Value

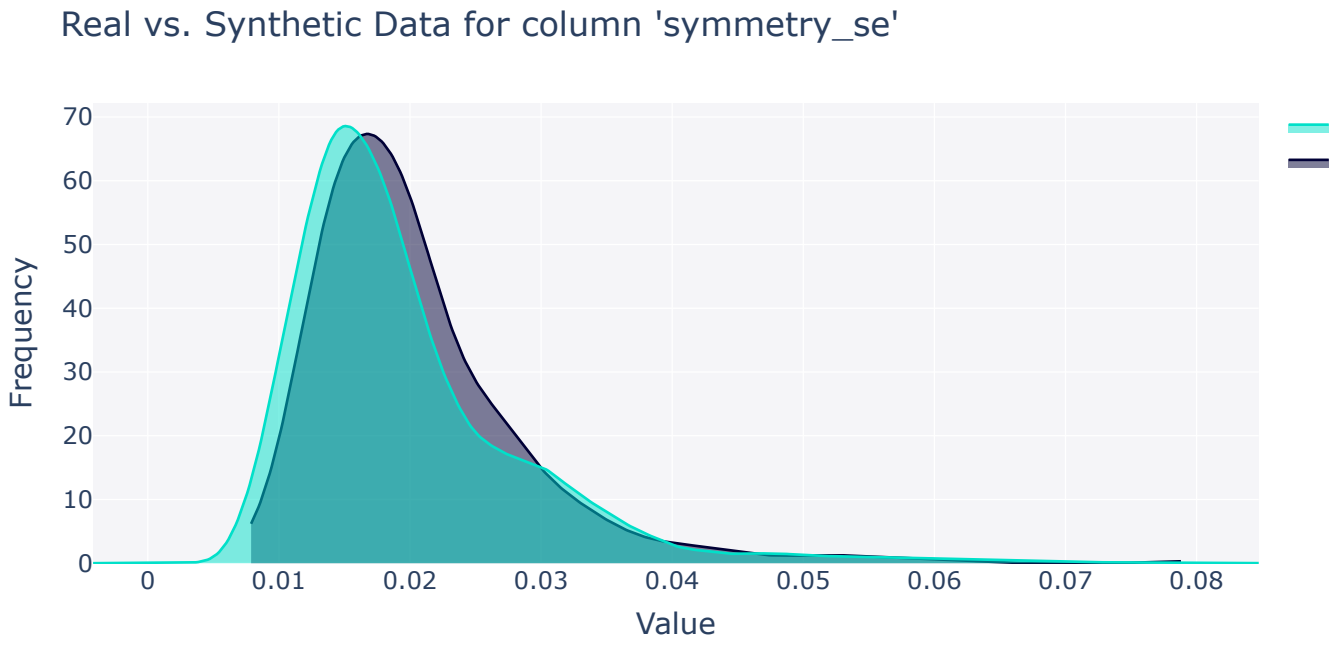

Real vs. Synthetic Data for column 'area_se'
Frequency
0.03
0.025
0.02
0.015
0.01
0.005
0
0
100
200
300
400
500
600
700
800
900
Value

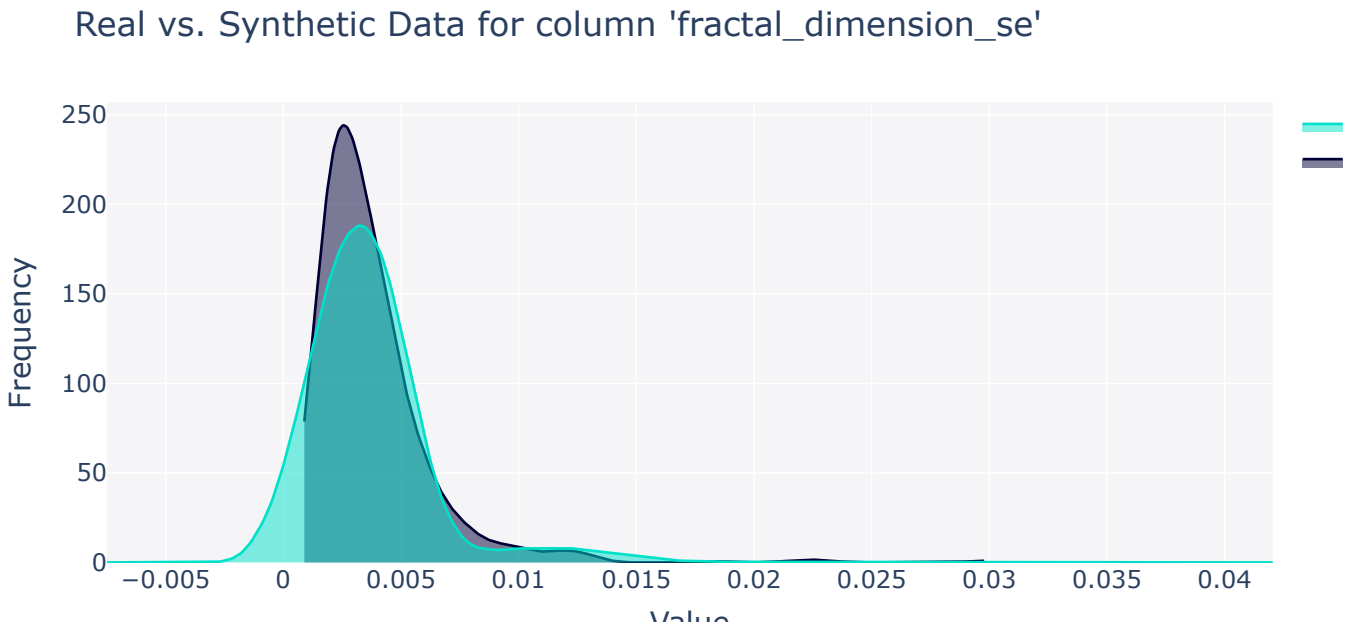

Real vs. Synthetic Data for column 'smoothness_se'
Frequency
150
100
50
0
0
0.005
0.01
0.015
0.02
0.025
0.03
0.035
Value

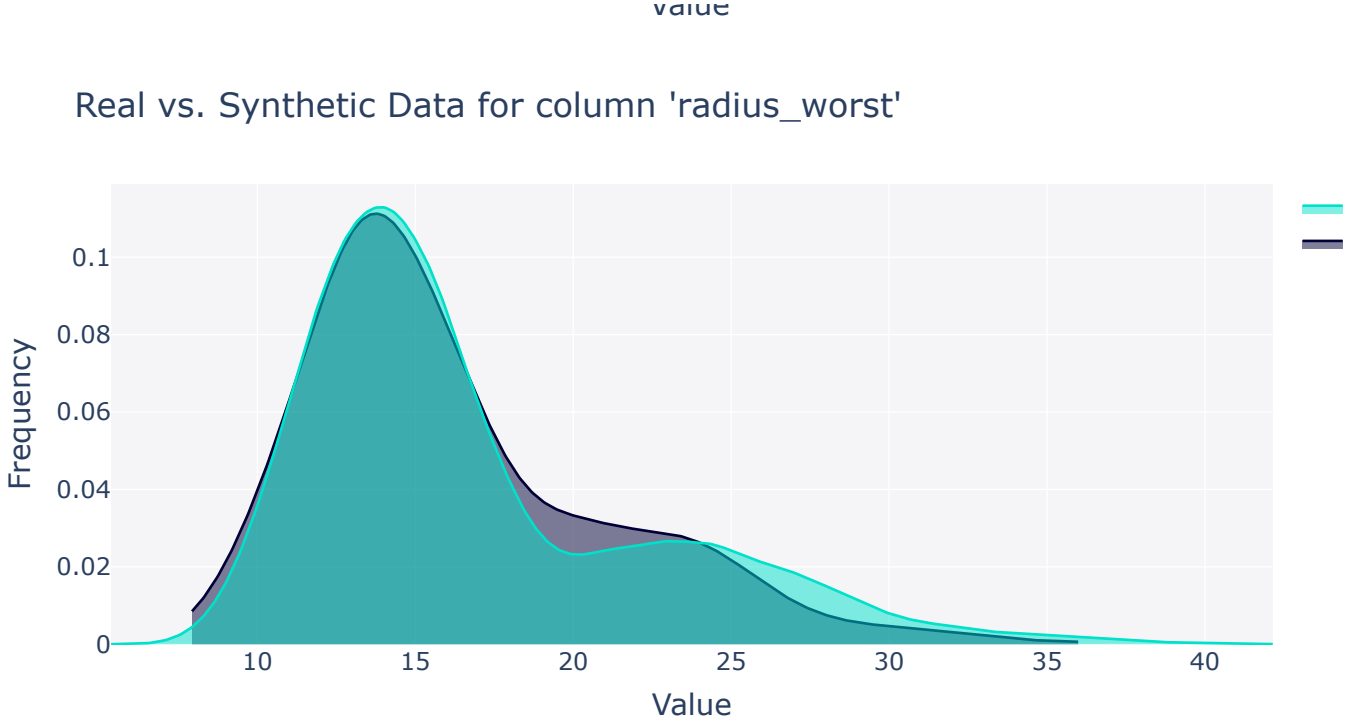

Real vs. Synthetic Data for column 'compactness_se'
Frequency
35
30
25
20
15
10
5
0
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
Value

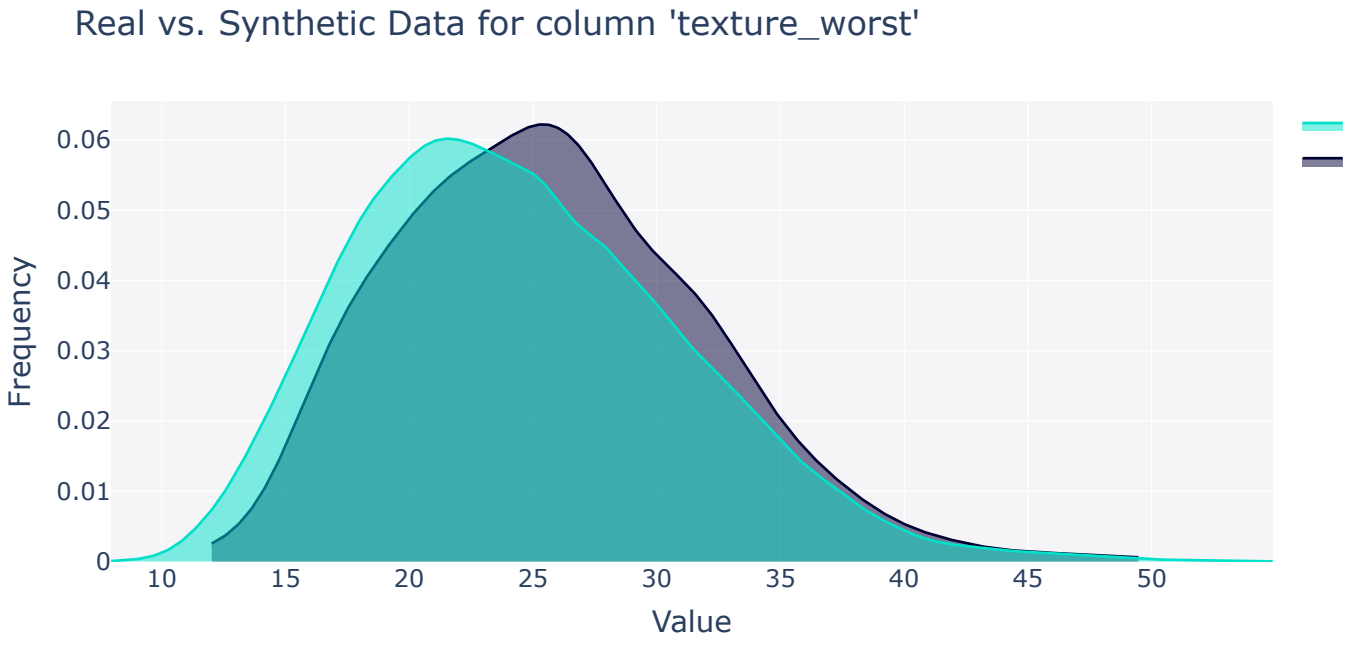

Real vs. Synthetic Data for column 'concavity_se'
Frequency
20
15
10
5
0
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
Value

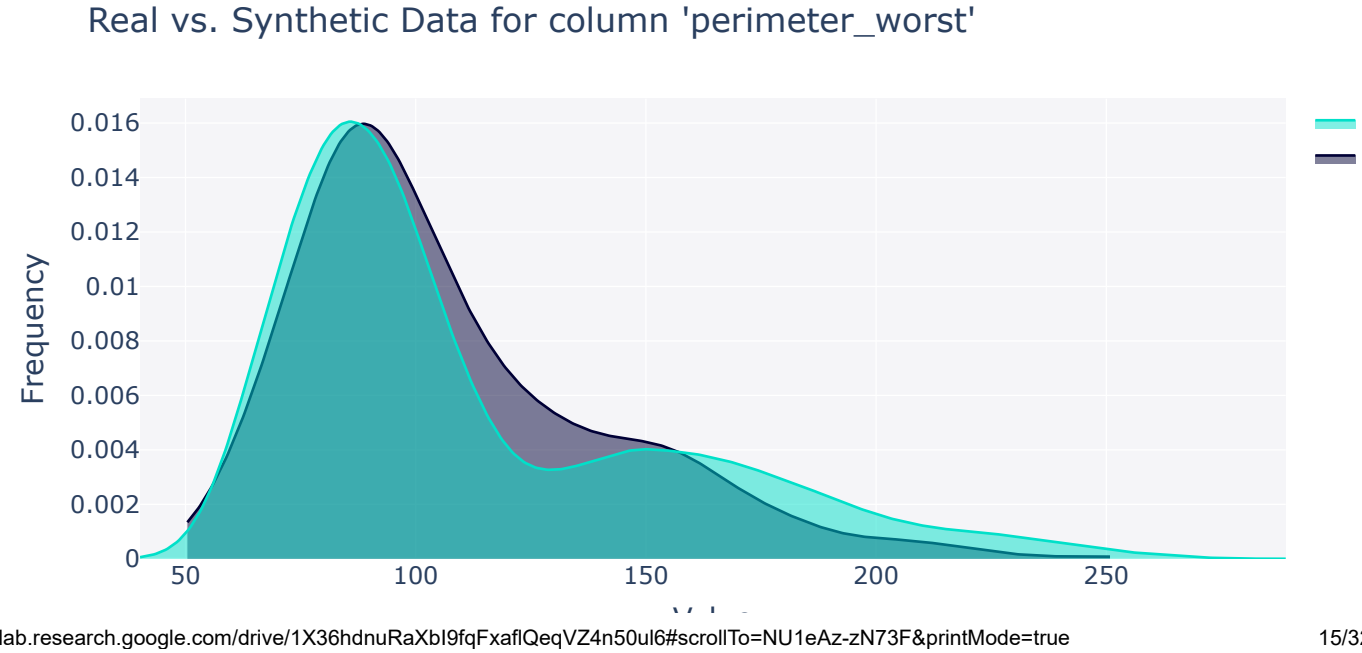

Real vs. Synthetic Data for column 'concave points_se'
Frequency
Value
0
10
20
30
40
50
60
70
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07


Real vs. Synthetic Data for column 'symmetry_se'
Frequency
Value
0
10
20
30
40
50
60
70
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08

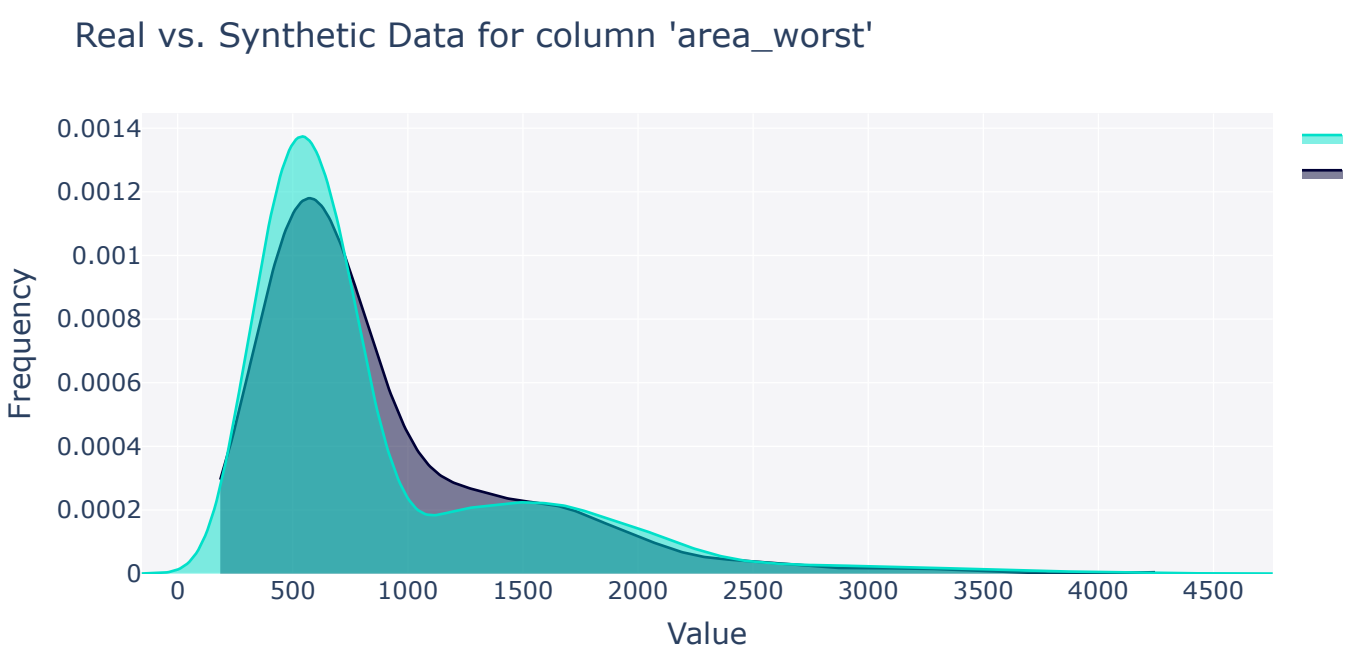

Real vs. Synthetic Data for column 'fractal_dimension_se'
Frequency
0
50
100
150
200
250
−0.005
0
0.005
0.01
0.015
0.02
0.025
0.03
0.035
0.04

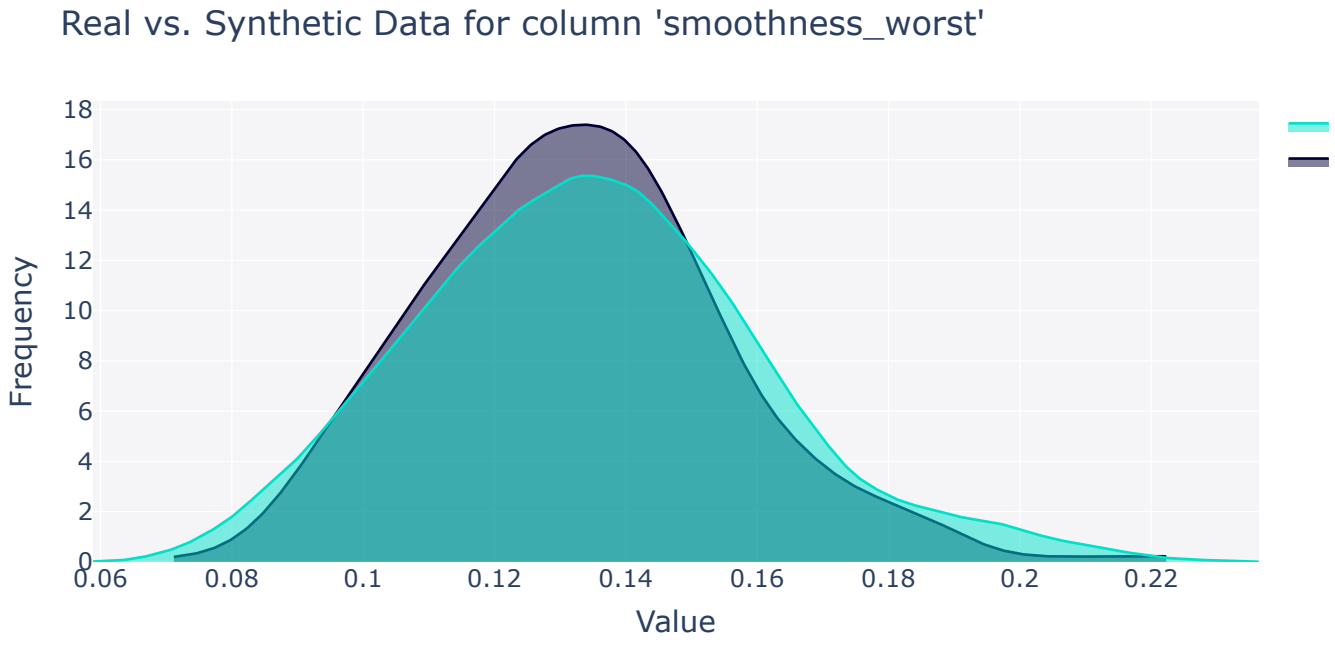

Real vs. Synthetic Data for column 'radius_worst'
Frequency
0.1
0.08
0.06
0.04
0.02
0
10
15
20
25
30
35
40
Value

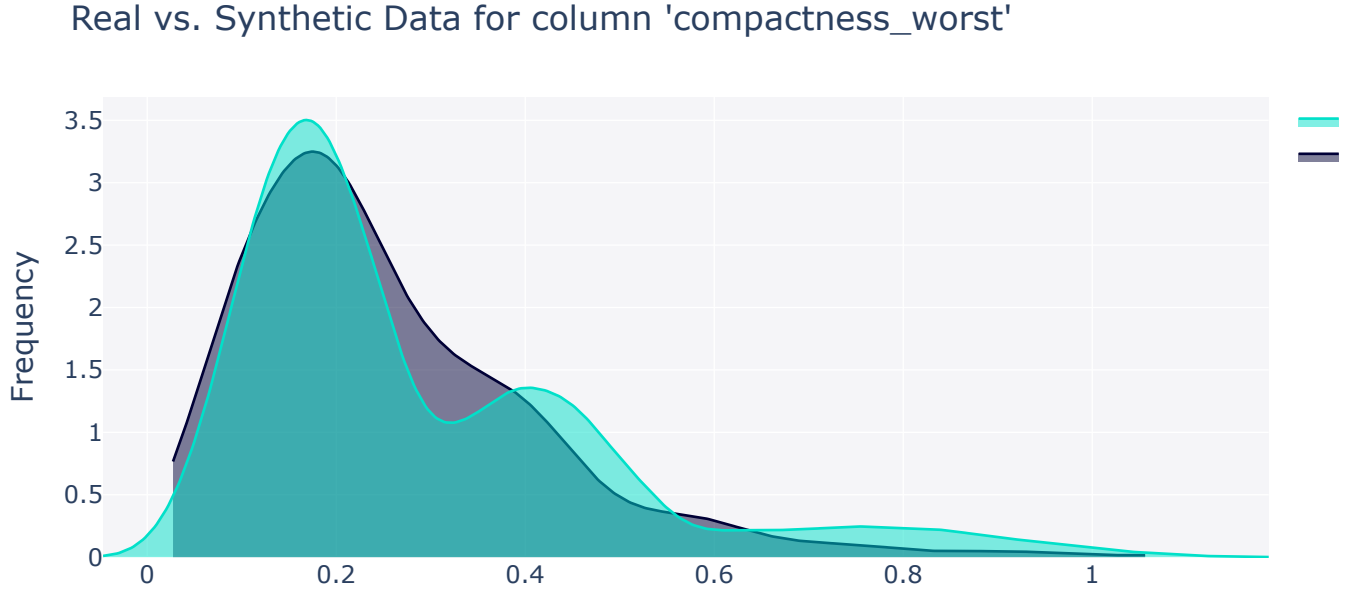

Real vs. Synthetic Data for column 'texture_worst'
Frequency
0.06
0.05
0.04
0.03
0.02
0.01
0
10
15
20
25
30
35
40
45
50
Value


Real vs. Synthetic Data for column 'perimeter_worst'
Frequency
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
50
100
150
200
250

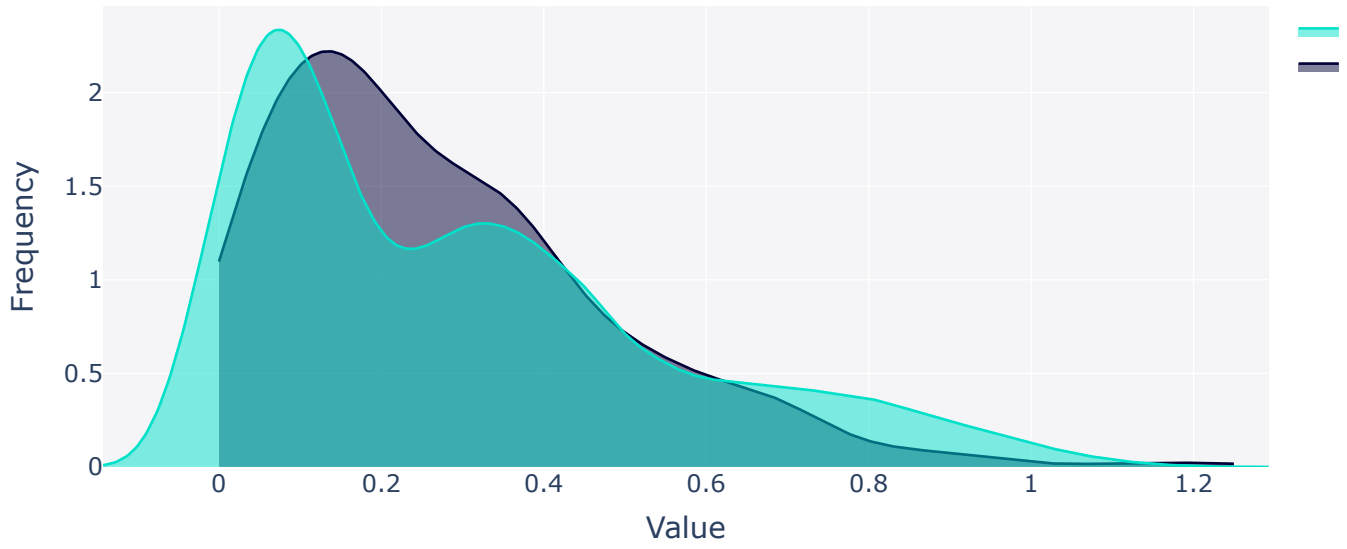
value

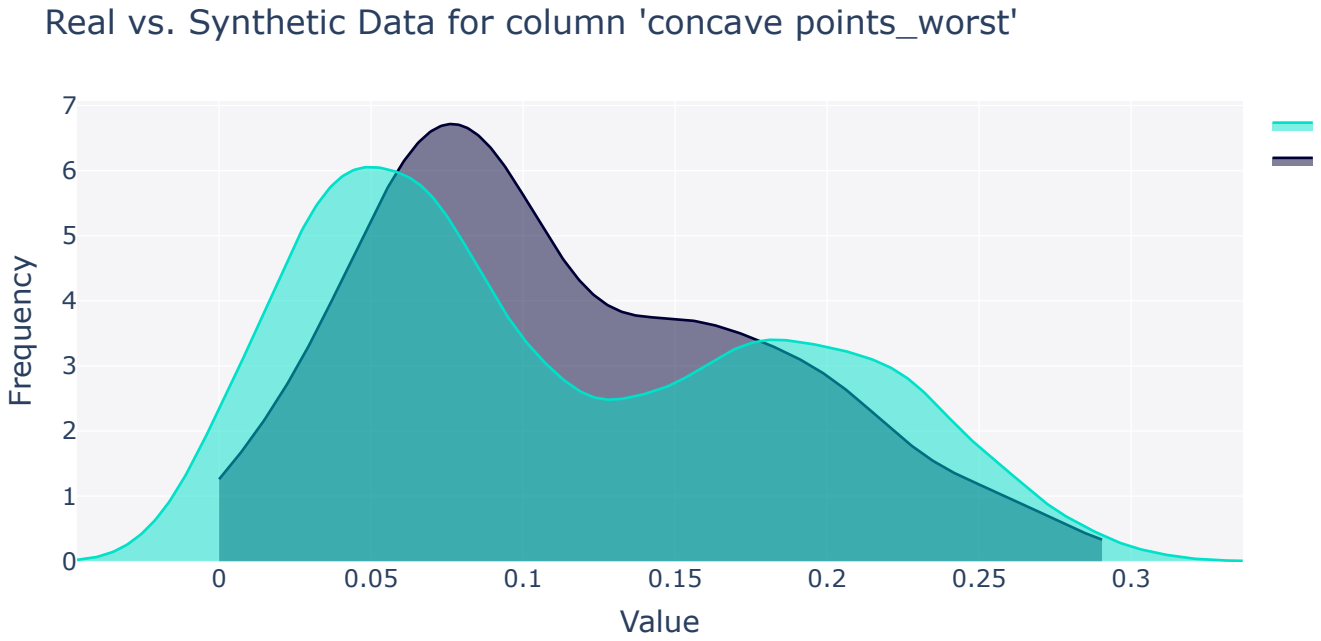

Real vs. Synthetic Data for column 'area_worst'
Frequency
0.0014
0.0012
0.001
0.0008
0.0006
0.0004
0.0002
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
Value

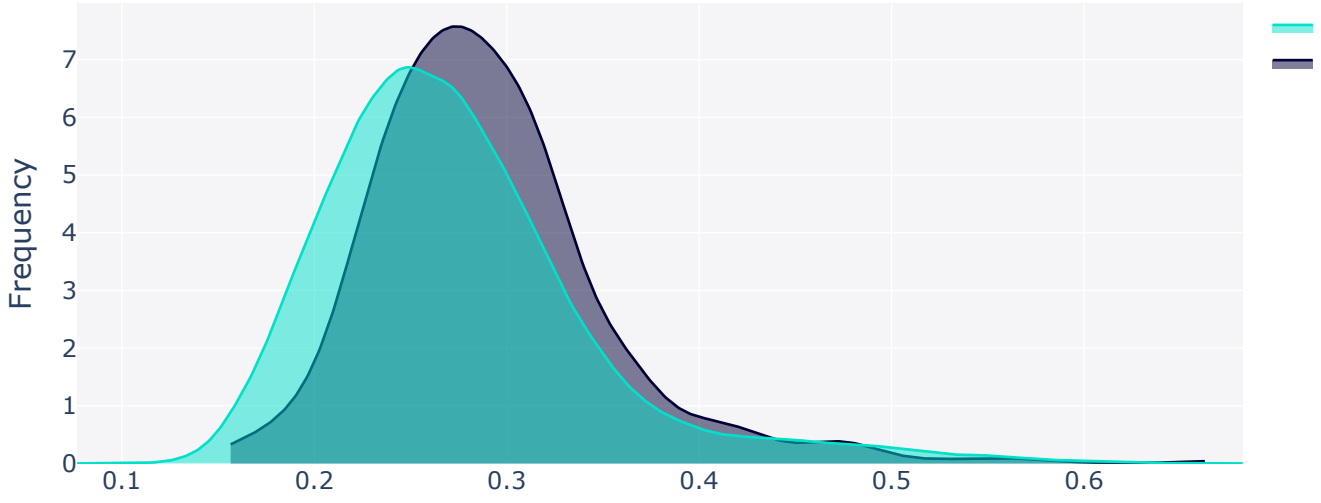

Real vs. Synthetic Data for column 'smoothness_worst'
Frequency
18
16
14
12
10
8
6
4
2
0
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2
0.22
Value

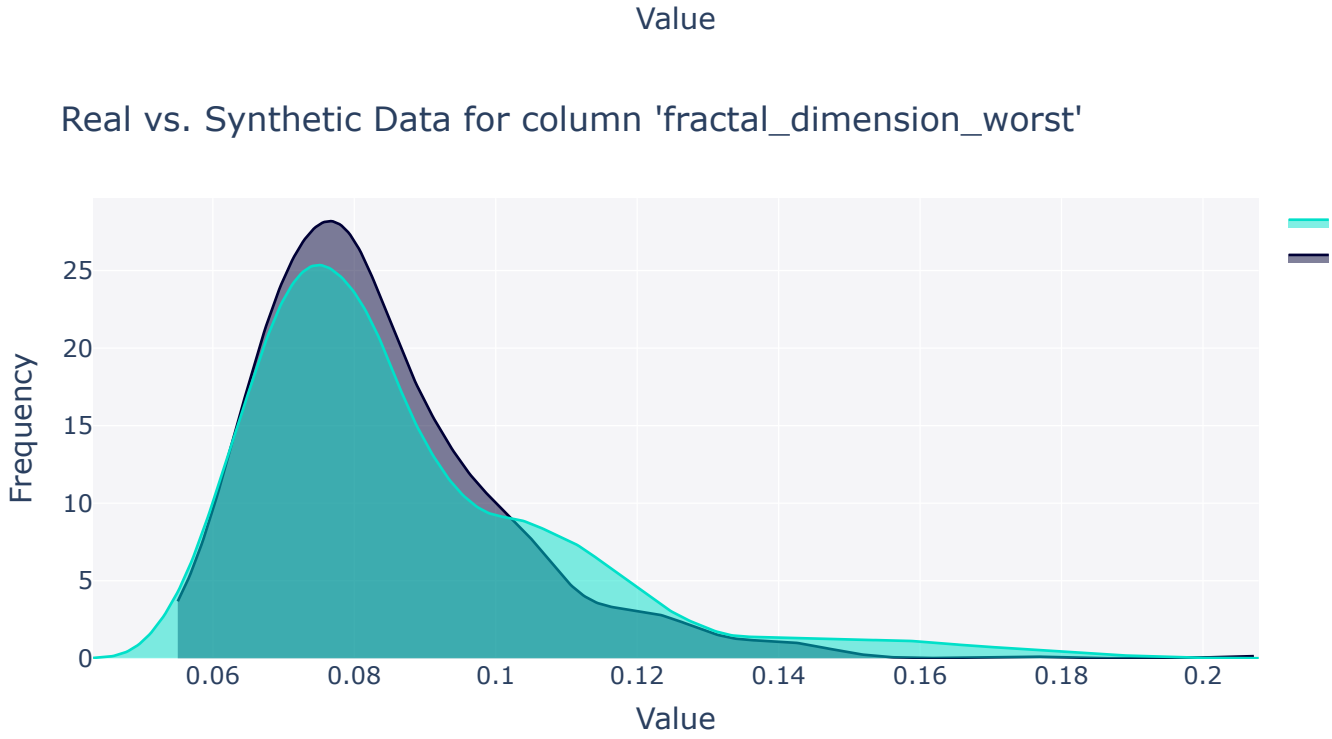

Real vs. Synthetic Data for column 'compactness_worst'
Frequency
3.5
3
2.5
2
1.5
1
0.5
0
0
0.2
0.4
0.6
0.8
1

Value


Real vs. Synthetic Data for column 'concavity_worst'
Frequency
0
0.5
1
1.5
2
0
0.2
0.4
0.6
0.8
1
1.2
Value


Real vs. Synthetic Data for column 'concave points_worst'
Frequency
0
1
2
3
4
5
6
7
0
0.05
0.1
0.15
0.2
0.25
0.3
Value


Real vs. Synthetic Data for column 'symmetry_worst'
Frequency
0
1
2
3
4
5
6
7
0.1
0.2
0.3
0.4
0.5
0.6

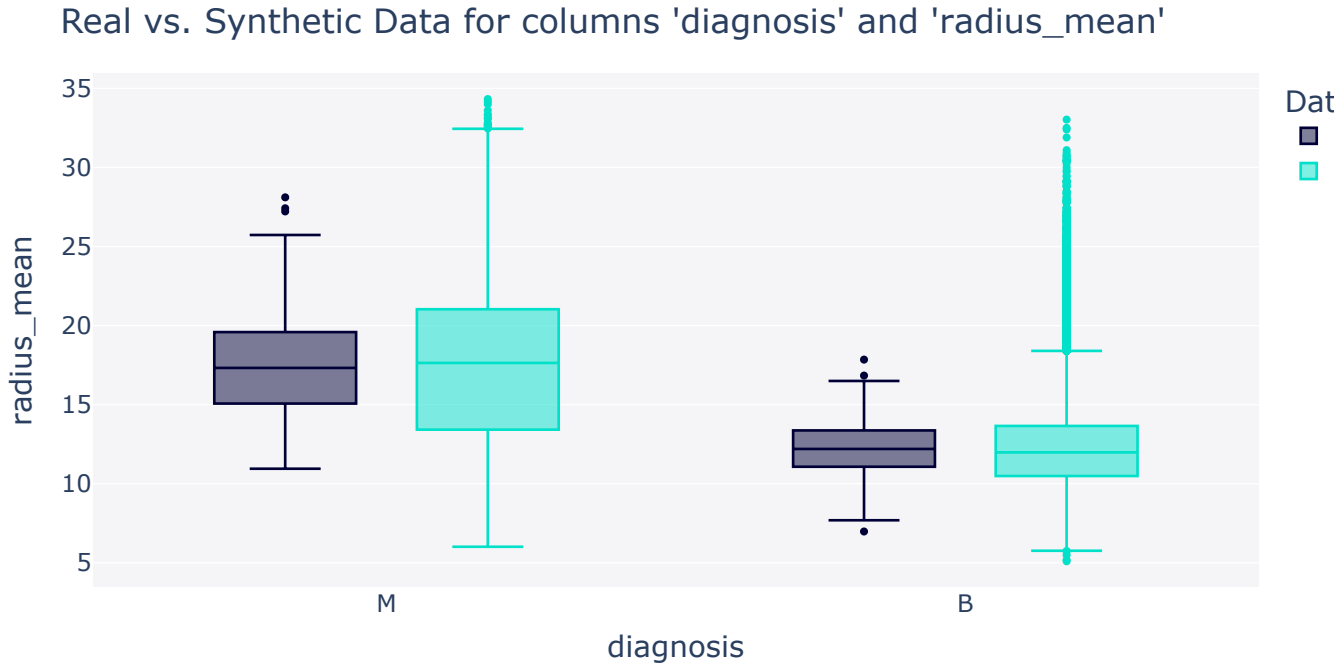


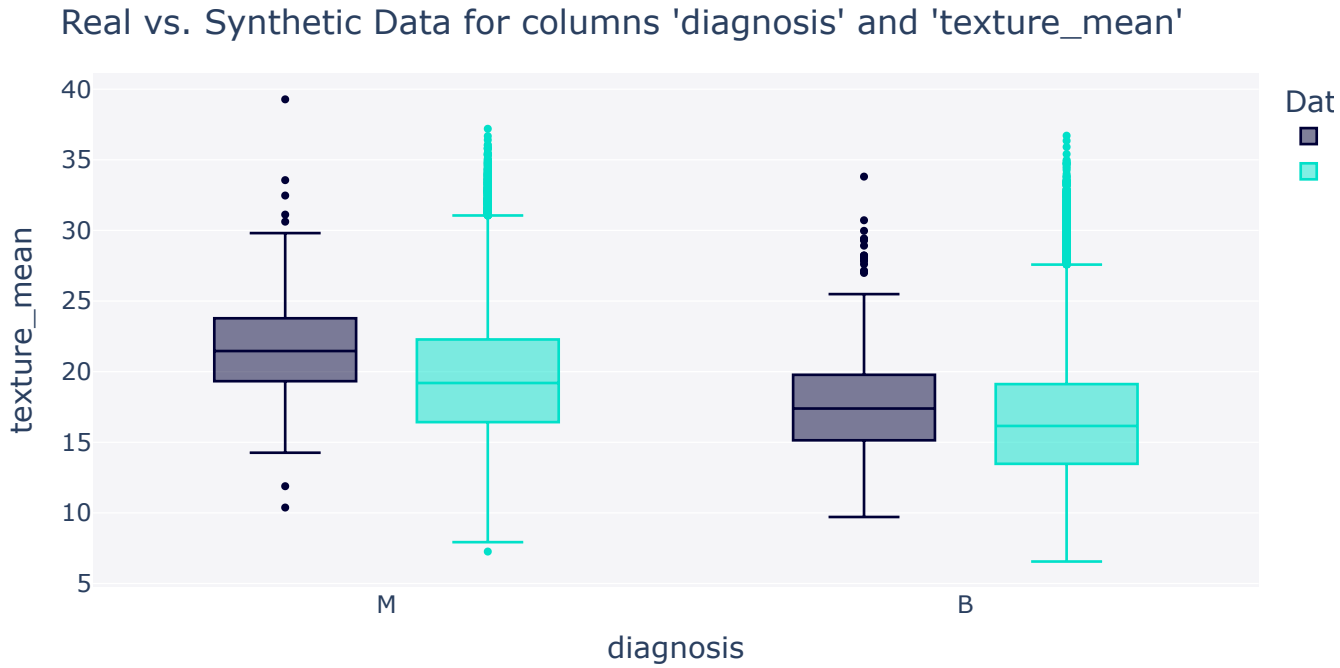

```
display(synthetic_data)
```

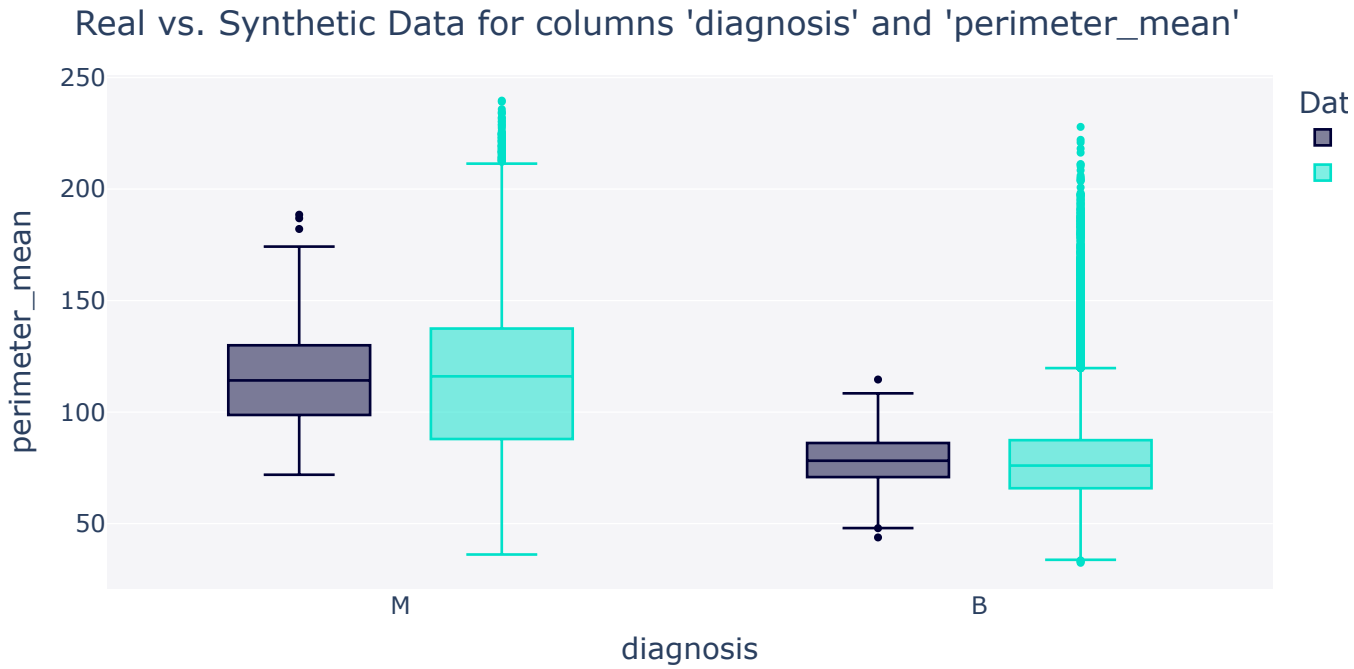
	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	con… points_…
0	B	9.729198	25.045711	87.490998	469.630694	0.118687	0.084707	0.095953	0.006
1	B	10.761181	15.357311	77.948740	358.555082	0.075049	0.160262	-0.003265	0.006
2	B	14.729835	13.781208	69.181171	240.953477	0.085235	0.127278	0.046510	0.022
3	M	11.041602	15.166336	79.590092	487.758194	0.088066	0.122436	0.117555	0.025
4	M	22.212884	23.750215	147.834552	1356.086418	0.078053	0.070602	0.063281	0.029
...	...	...	...	...	...	...	...	...	
99995	M	11.539701	9.774781	84.175863	272.360627	0.086093	0.053266	0.035945	0.008
99996	M	14.318422	22.072453	195.407267	676.396665	0.110719	0.262314	0.393196	0.117
99997	M	12.710631	24.599054	86.630484	337.597984	0.116655	0.251271	0.397309	0.044
99998	M	23.890062	26.008675	158.215124	1928.849538	0.113463	0.262159	0.249422	0.096
99999	B	13.355747	17.926245	85.885990	350.570989	0.074397	0.047172	-0.011635	0.001

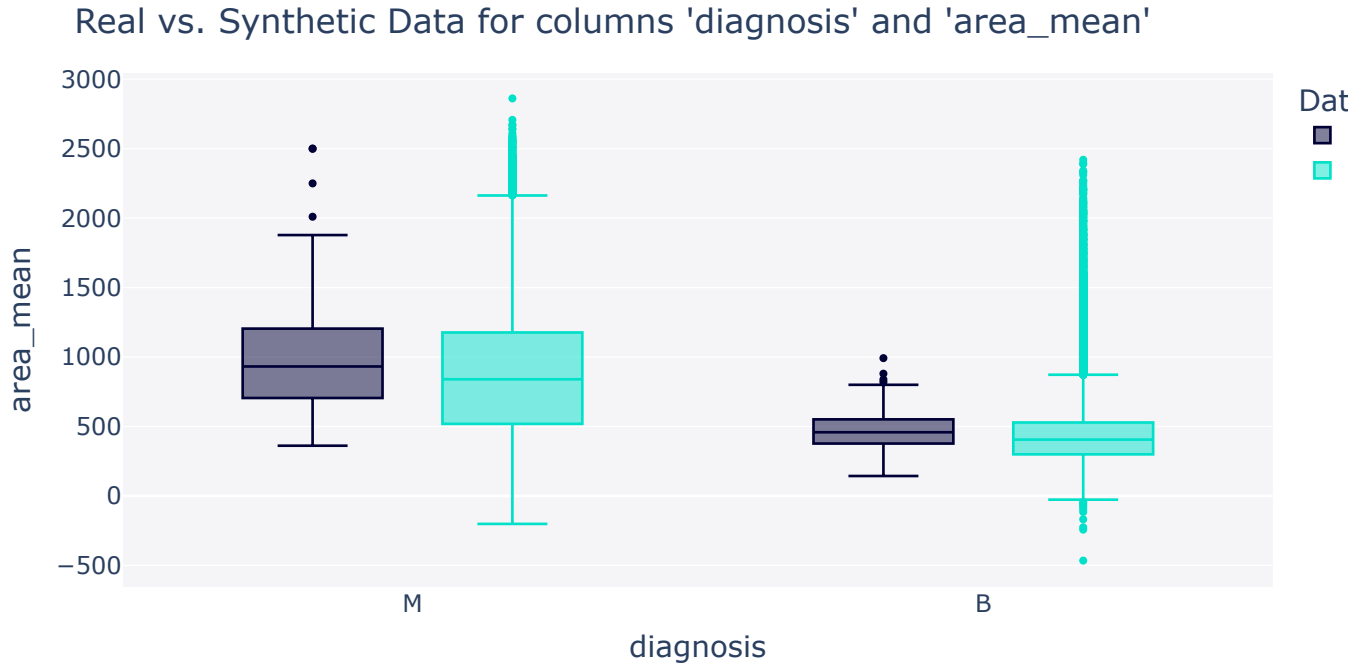
100000 rows × 31 columns

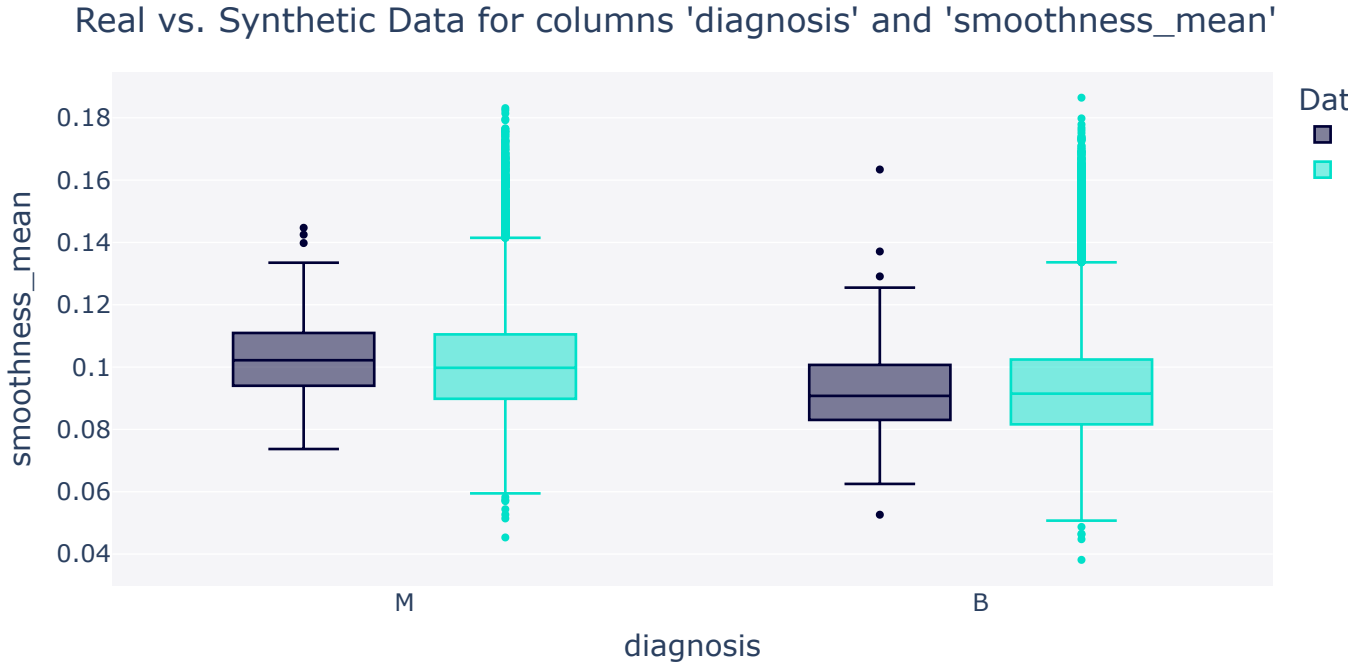
```
from itertools import combinations
from matplotlib.backends.backend_pdf import PdfPages

# Get all column pairs
column_pairs = combinations(df.columns, 2)

# Loop through each column pair
for column1, column2 in column_pairs:
  # Generate the plot using get_column_pair_plot
  fig = get_column_pair_plot(
      real_data=df,
      synthetic_data=synthetic_data,
      column_names=[column1, column2]
  )


  fig.show()
```

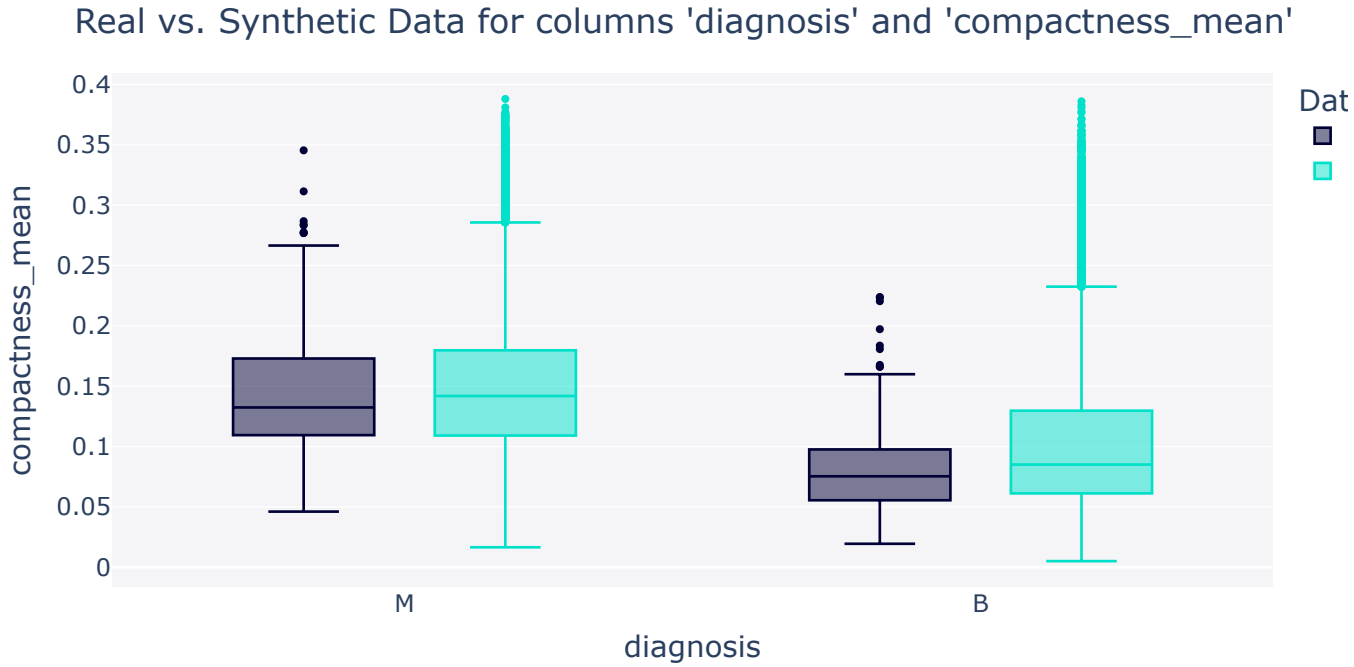

Real vs. Synthetic Data for columns 'diagnosis' and 'radius_mean'
radius_mean
35
30
25
20
15
10
5
M
B
diagnosis
Dat

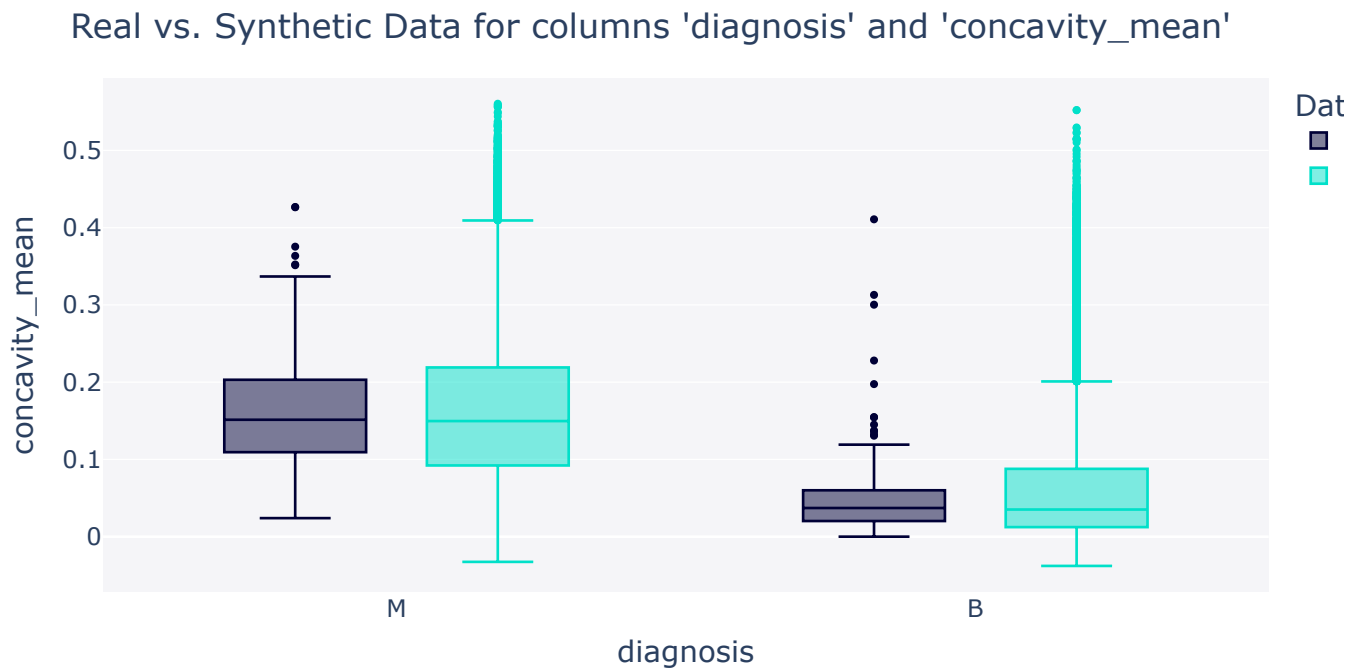

Real vs. Synthetic Data for columns 'diagnosis' and 'texture_mean'
texture_mean
40
35
30
25
20
15
10
5
M
B
diagnosis
Dat

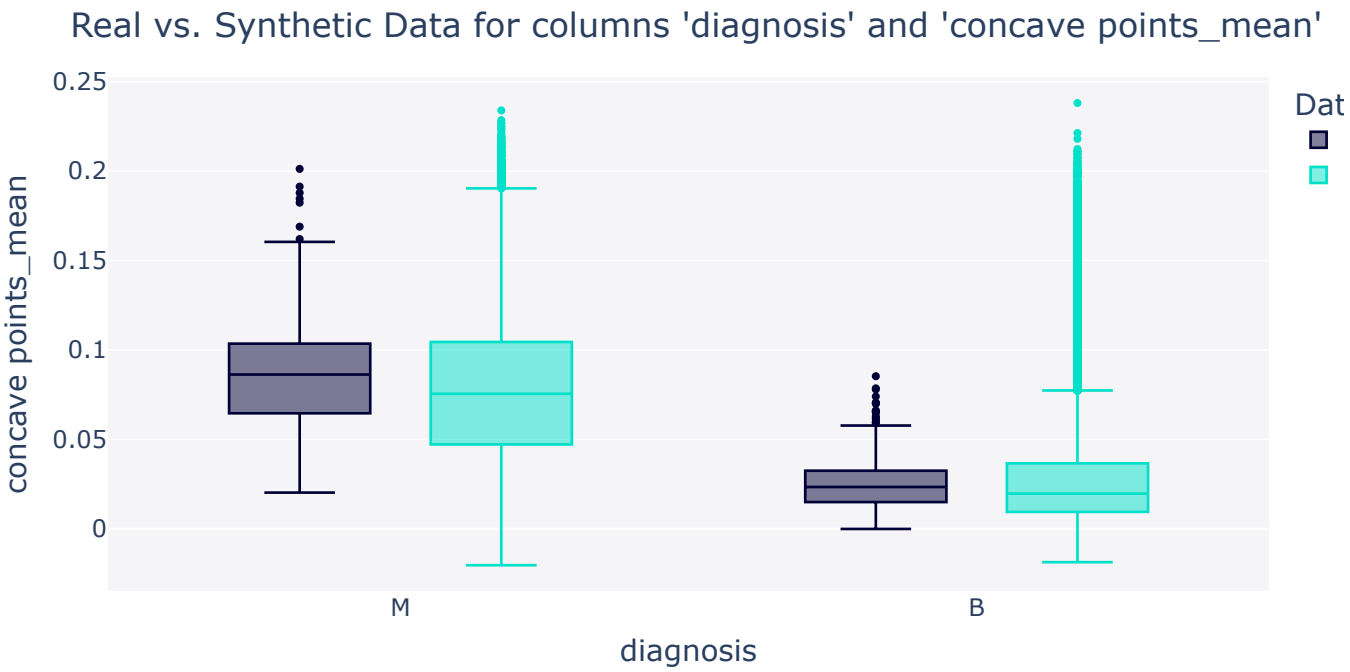

Real vs. Synthetic Data for columns 'diagnosis' and 'perimeter_mean'
perimeter_mean
250
200
150
100
50
M
B
diagnosis
Dat

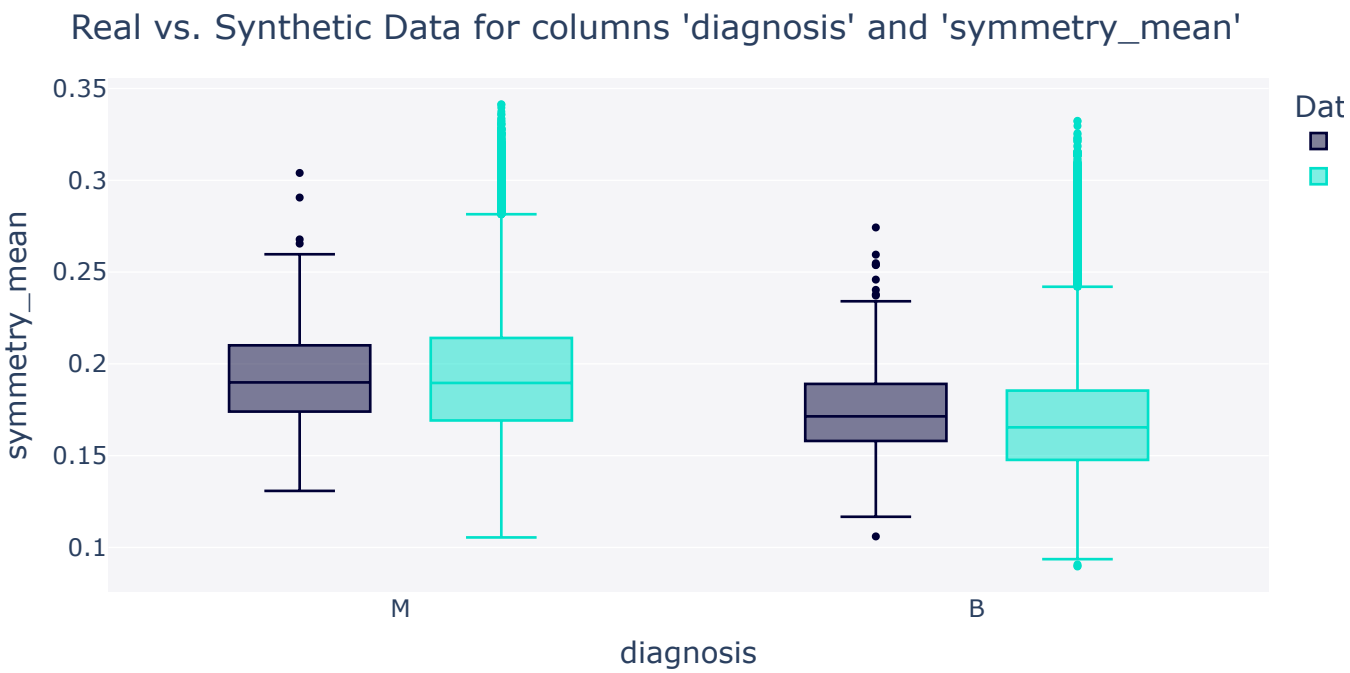

Real vs. Synthetic Data for columns 'diagnosis' and 'area_mean'
area_mean
3000
2500
2000
1500
1000
500
0
−500
M
B
diagnosis
Dat

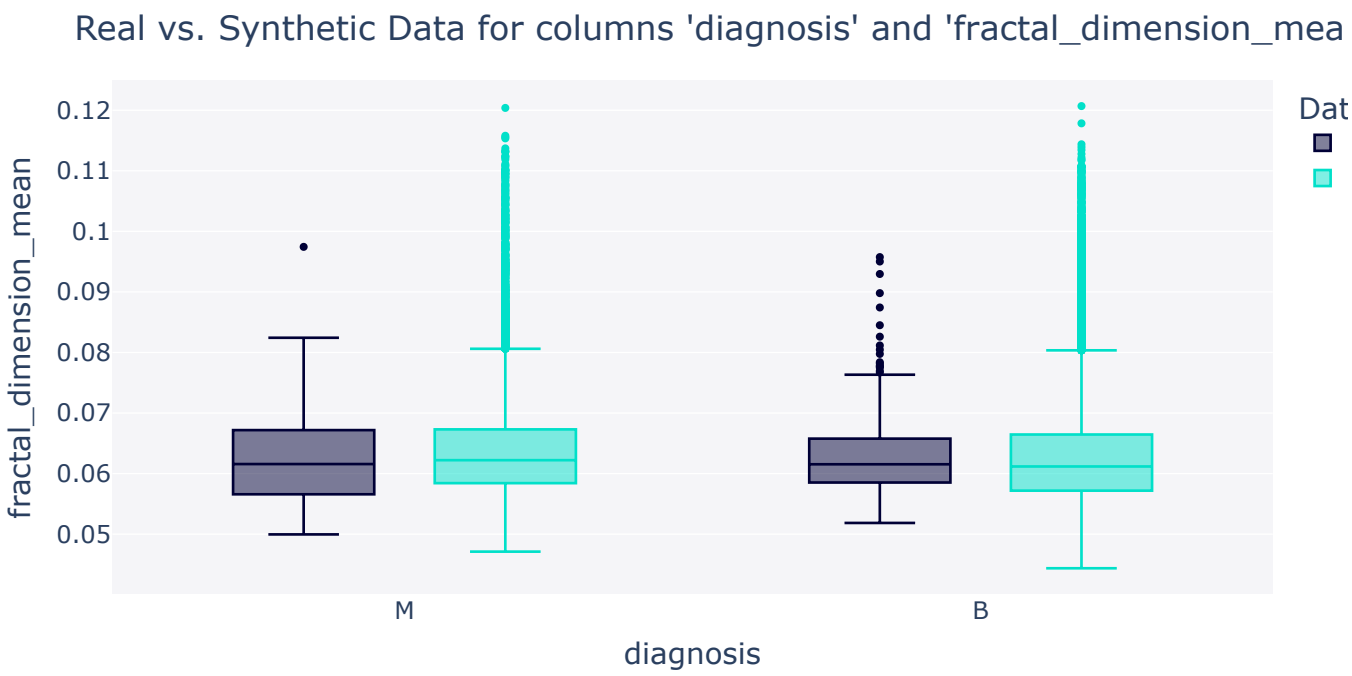

Real vs. Synthetic Data for columns 'diagnosis' and 'smoothness_mean'
smoothness_mean
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
M
B
diagnosis
Dat

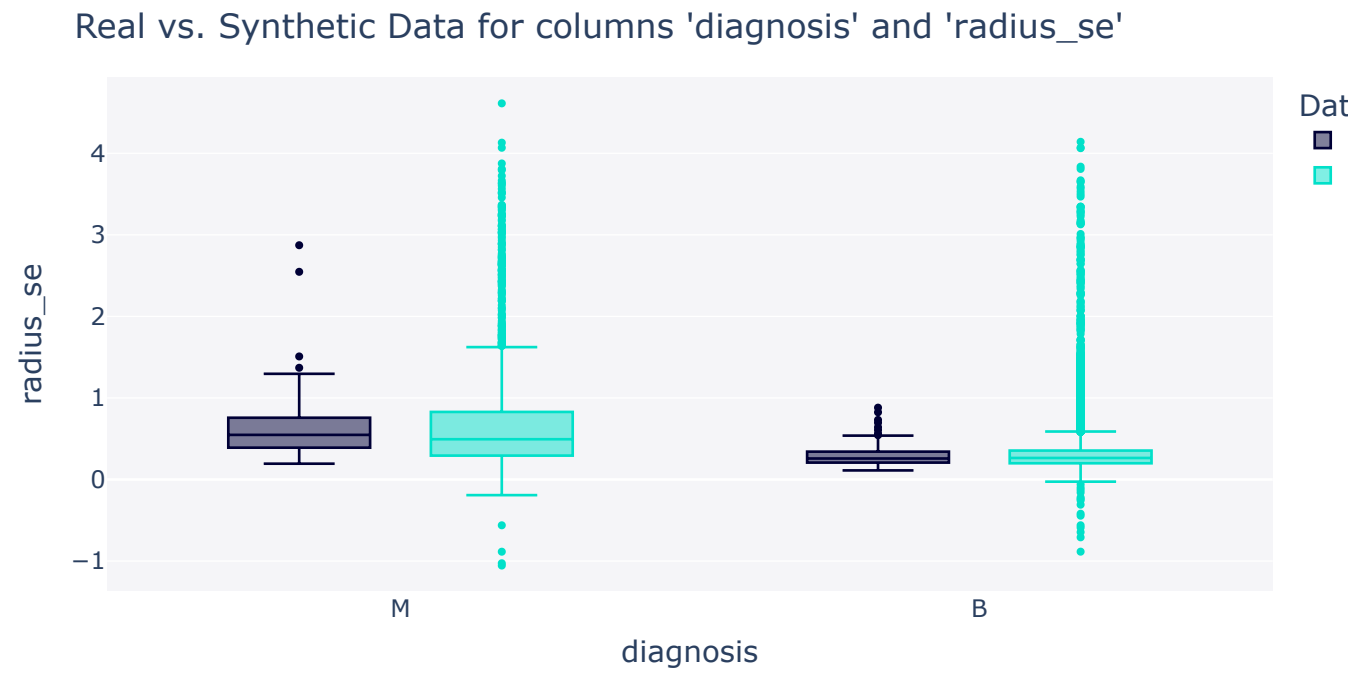

Real vs. Synthetic Data for columns 'diagnosis' and 'compactness_mean'
compactness_mean
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
M
B
diagnosis
Dat

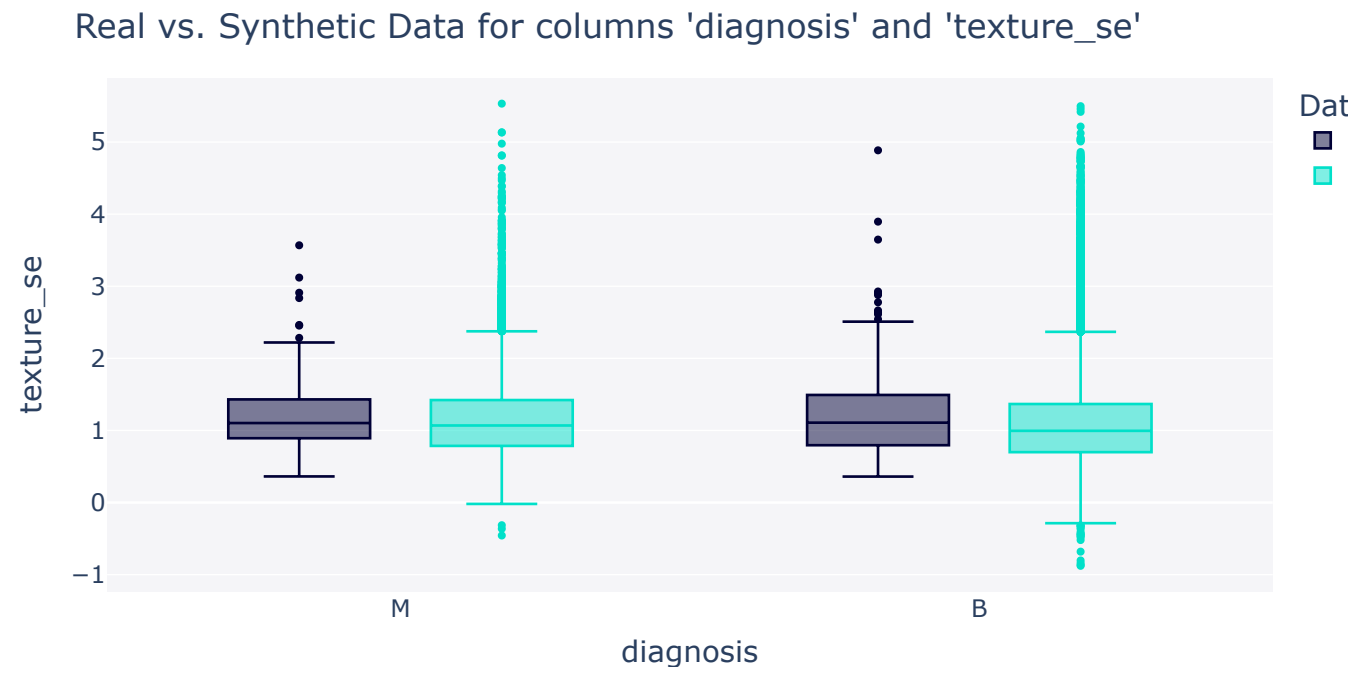

Real vs. Synthetic Data for columns 'diagnosis' and 'concavity_mean'
concavity_mean
0.5
0.4
0.3
0.2
0.1
0
M
B
diagnosis
Dat

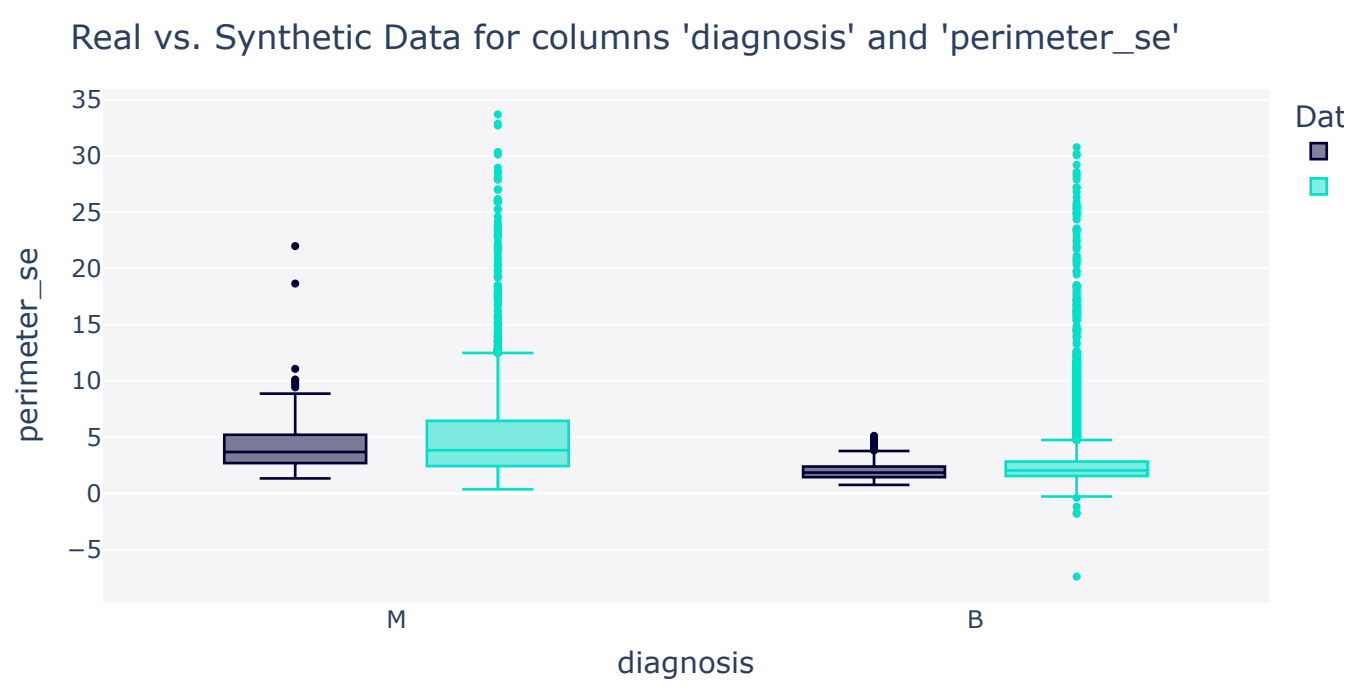

Real vs. Synthetic Data for columns 'diagnosis' and 'concave points_mean'
concave points_mean
0.25
0.2
0.15
0.1
0.05
0
M
B
diagnosis
Dat

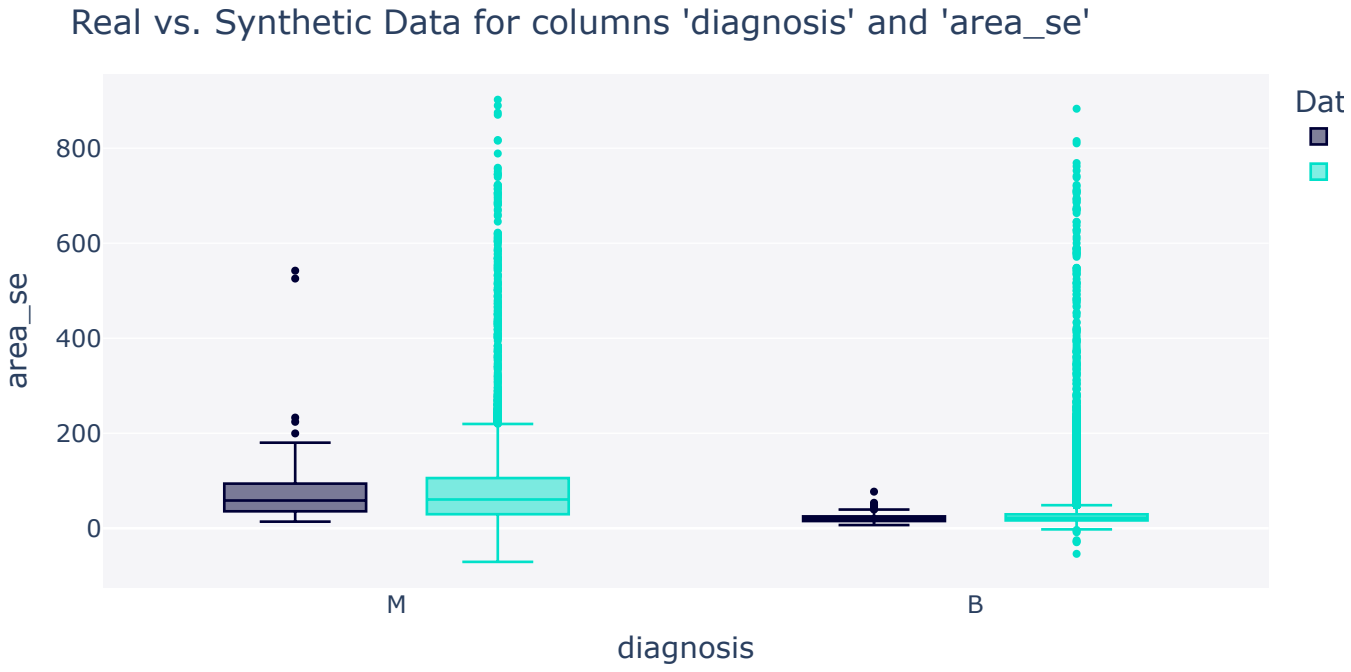

Real vs. Synthetic Data for columns 'diagnosis' and 'symmetry_mean'
symmetry_mean
0.35
0.3
0.25
0.2
0.15
0.1
M
B
diagnosis
Dat

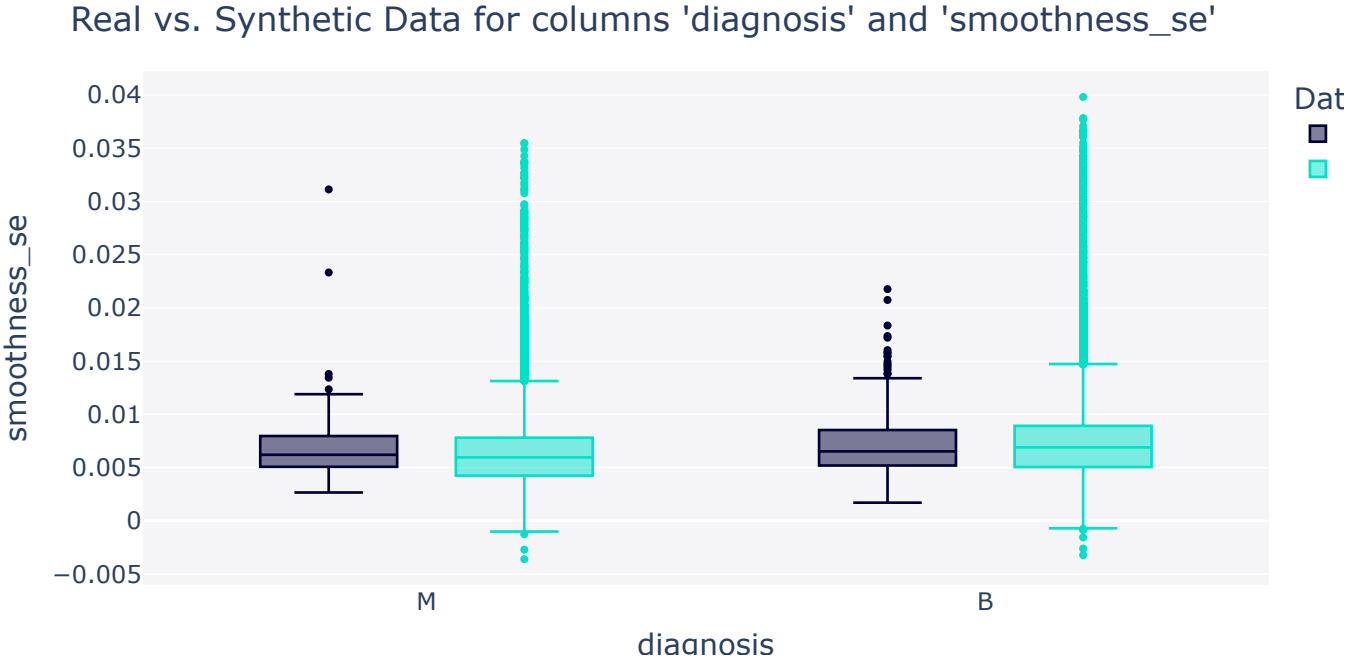

Real vs. Synthetic Data for columns 'diagnosis' and 'fractal_dimension_mea
fractal_dimension_mean
0.12
0.11
0.1
0.09
0.08
0.07
0.06
0.05
M
B
diagnosis
Dat

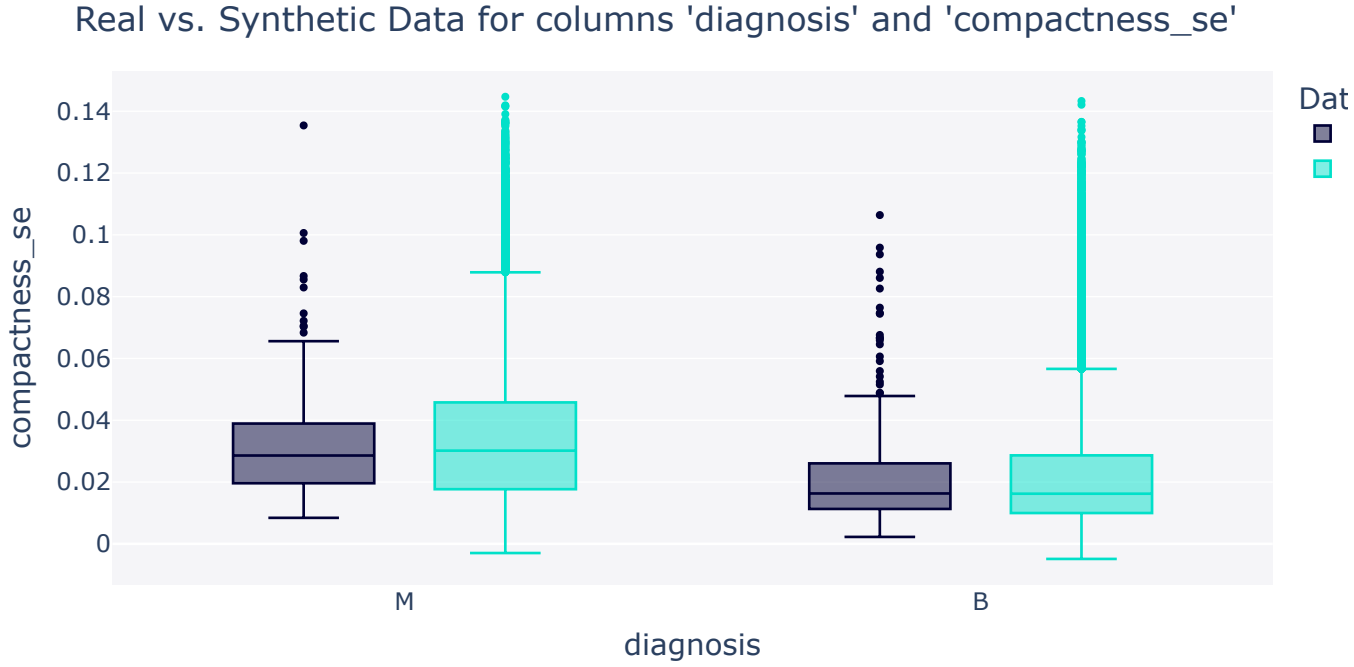

Real vs. Synthetic Data for columns 'diagnosis' and 'radius_se'
radius_se
4
3
2
1
0
−1
M
B
diagnosis
Dat

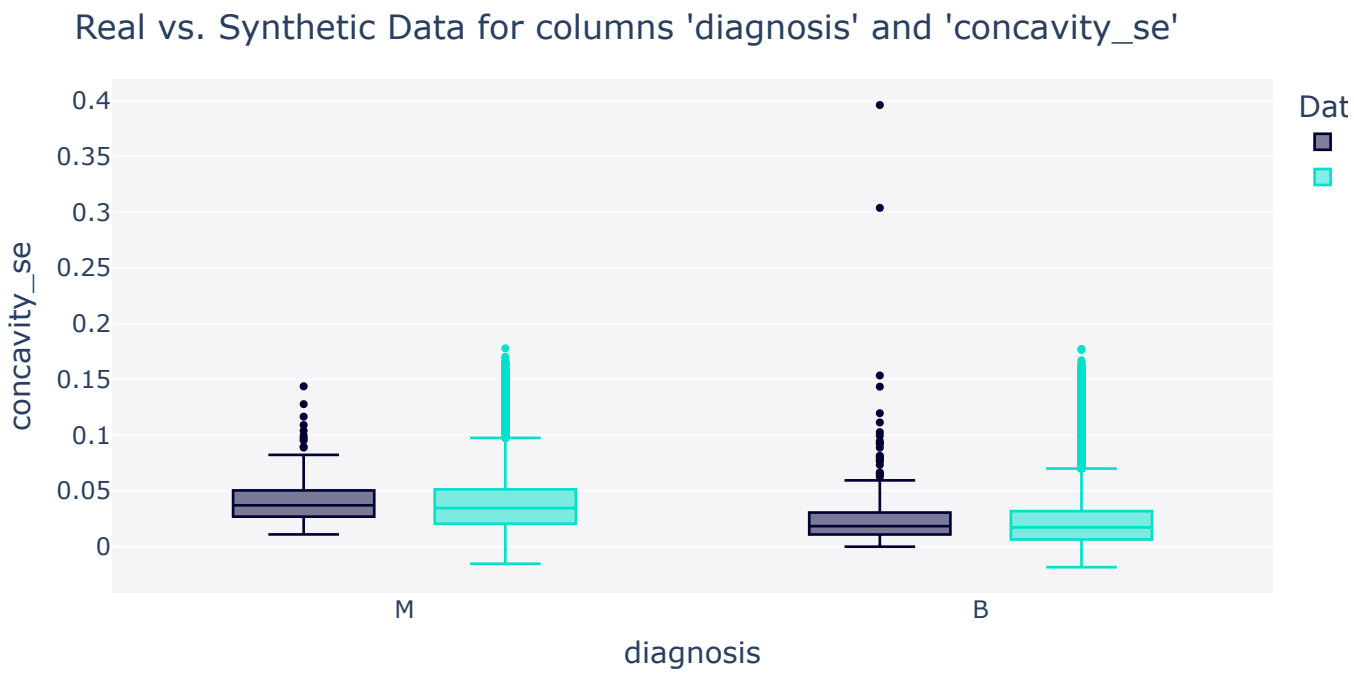

Real vs. Synthetic Data for columns 'diagnosis' and 'texture_se'
texture_se
5
4
3
2
1
0
−1
M
B
diagnosis
Dat

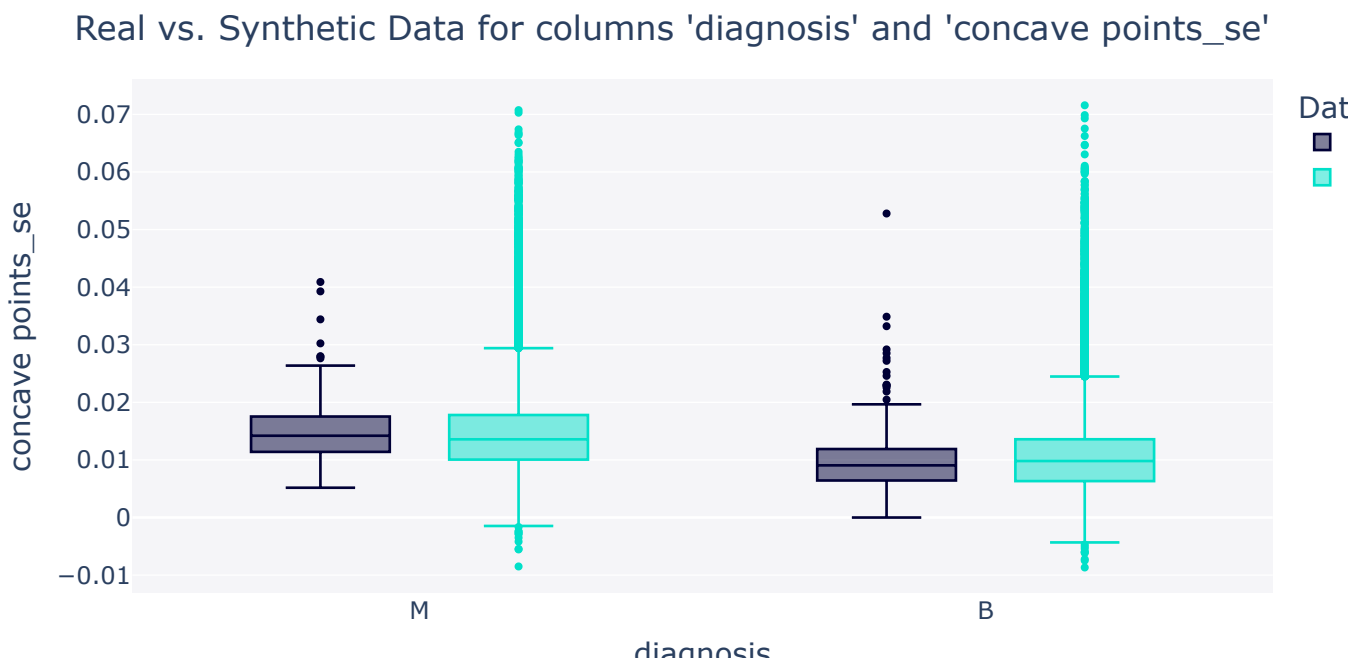

Real vs. Synthetic Data for columns 'diagnosis' and 'perimeter_se'
perimeter_se
35
30
25
20
15
10
5
0
−5
M
B
diagnosis
Dat

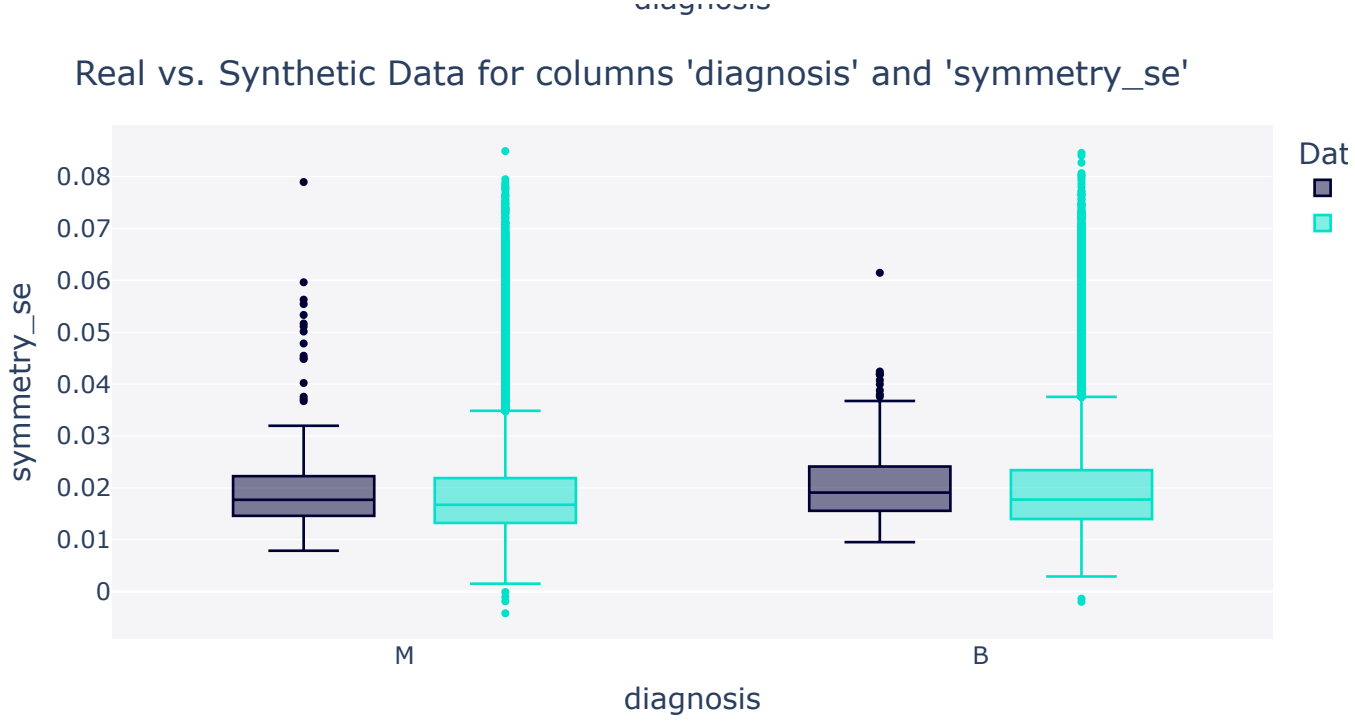

Real vs. Synthetic Data for columns 'diagnosis' and 'area_se'
area_se
800
600
400
200
0
M
B
diagnosis
Dat


Real vs. Synthetic Data for columns 'diagnosis' and 'smoothness_se'
smoothness_se
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
−0.005
M
B
diagnosis
Dat


Real vs. Synthetic Data for columns 'diagnosis' and 'compactness_se'
compactness_se
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
M
B
diagnosis
Dat

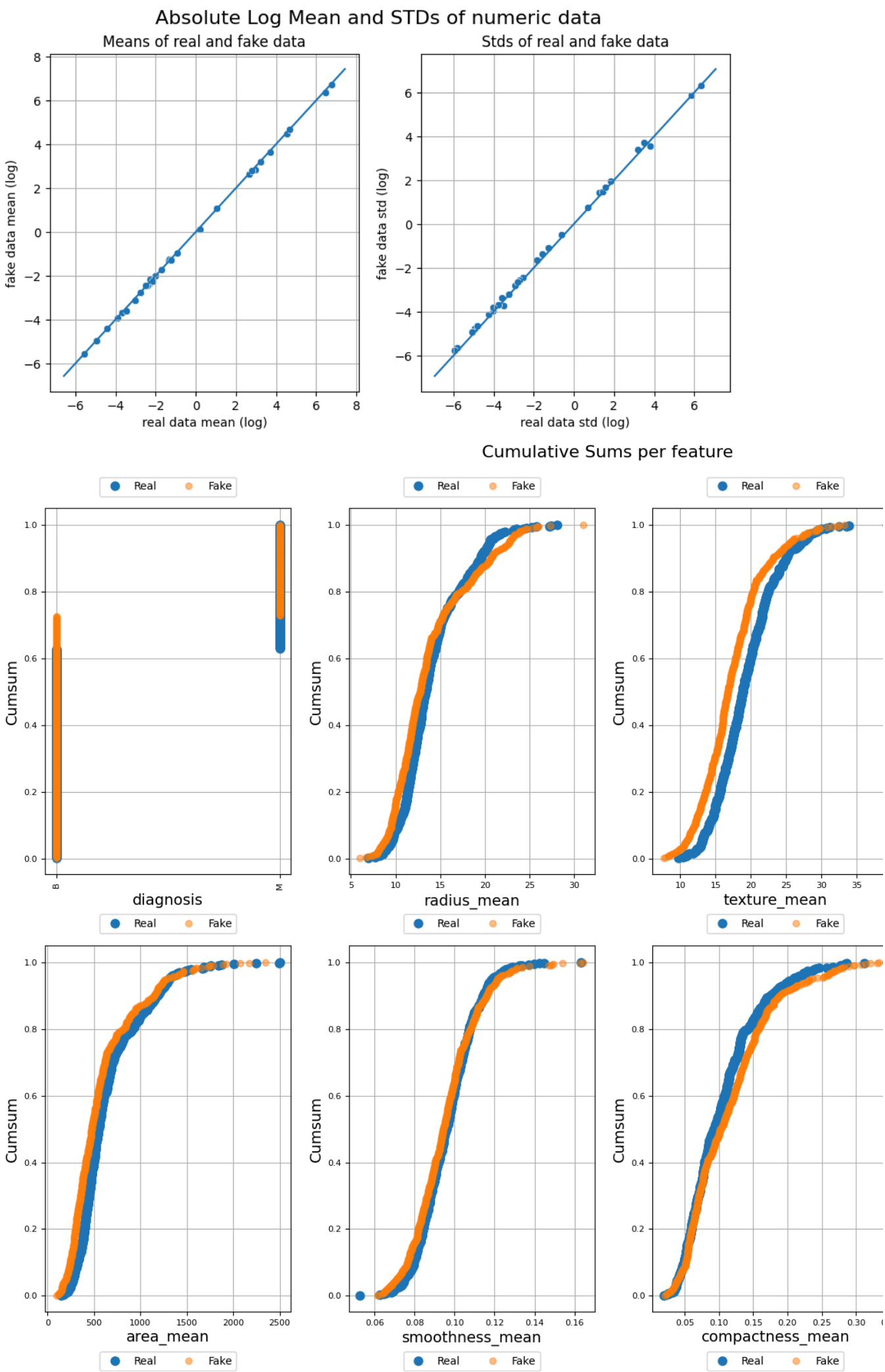

Real vs. Synthetic Data for columns 'diagnosis' and 'concavity_se'
concavity_se
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
M
B
diagnosis
Dat

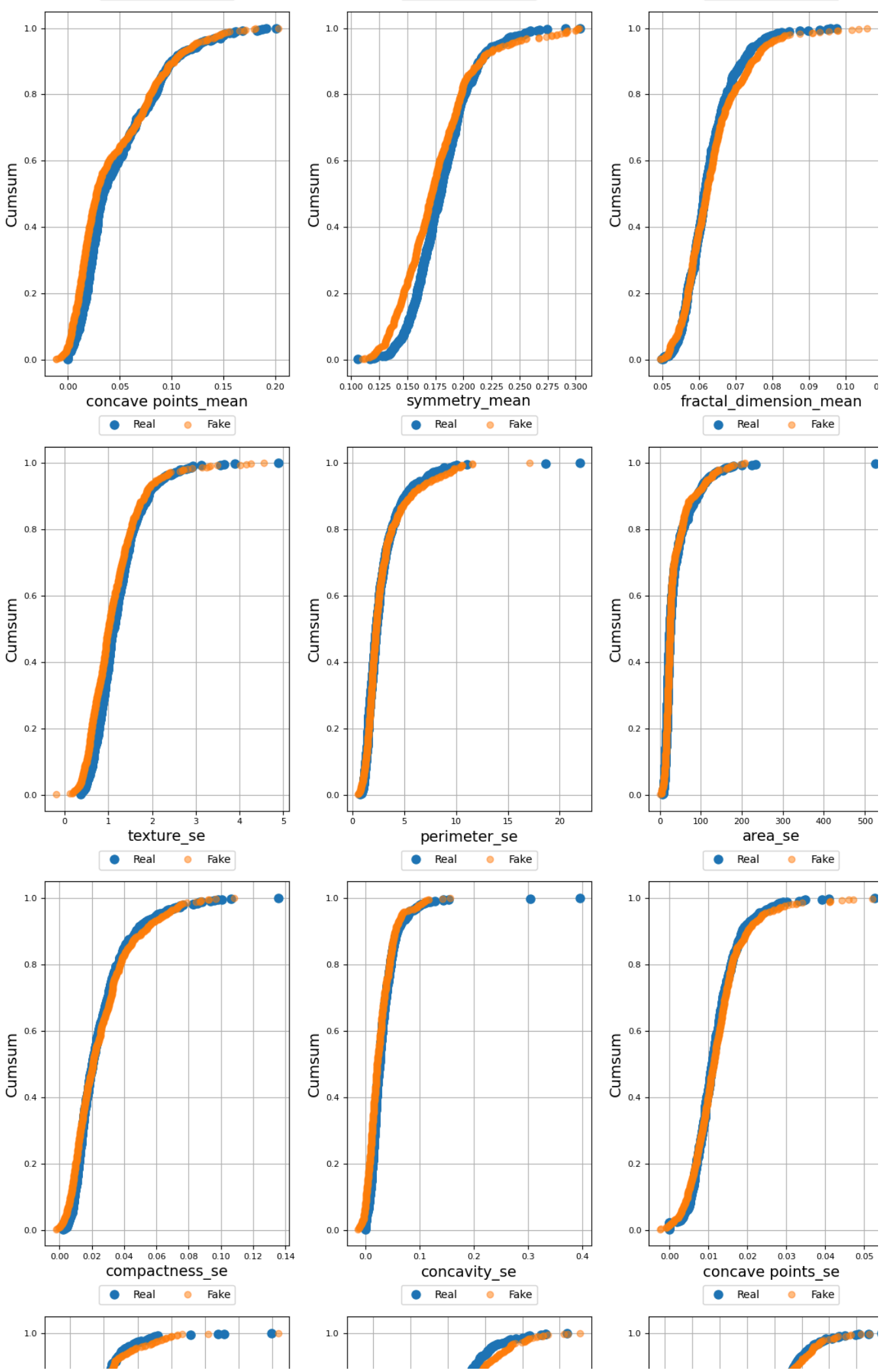

Real vs. Synthetic Data for columns 'diagnosis' and 'concave points_se'
concave points_se
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
−0.01
M
B
diagnosis
Dat

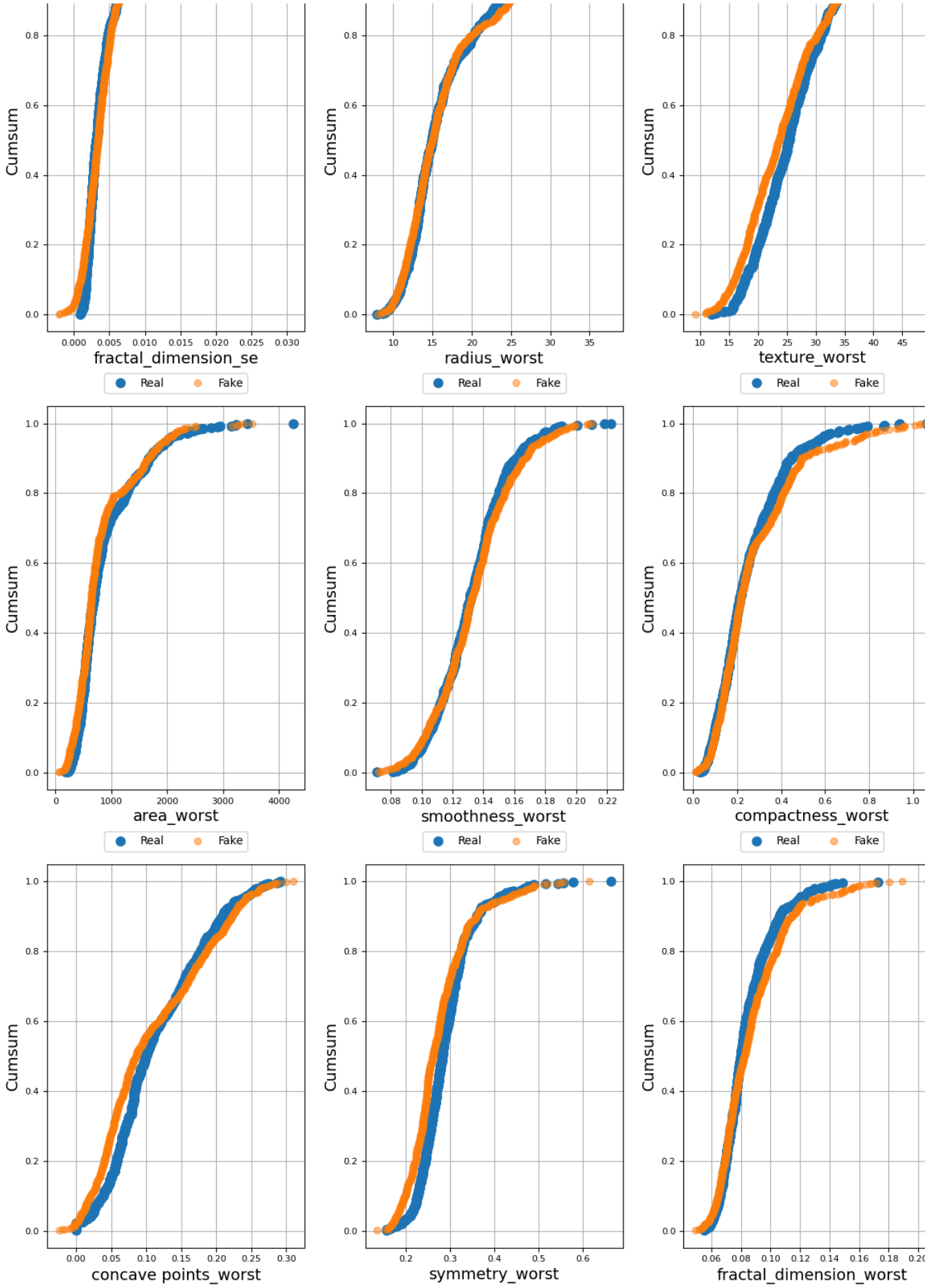



```
Buffered data was truncated after reaching the output size limit.
```

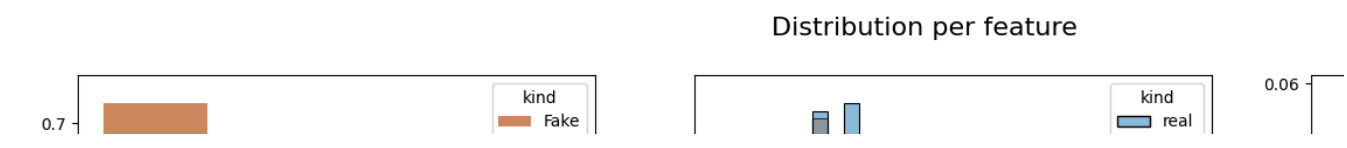
```
from table_evaluator import TableEvaluator

# Assuming real_data and synthetic_data are pandas DataFrames
table_evaluator = TableEvaluator(df, synthetic_data)

table_evaluator.visual_evaluation()
```

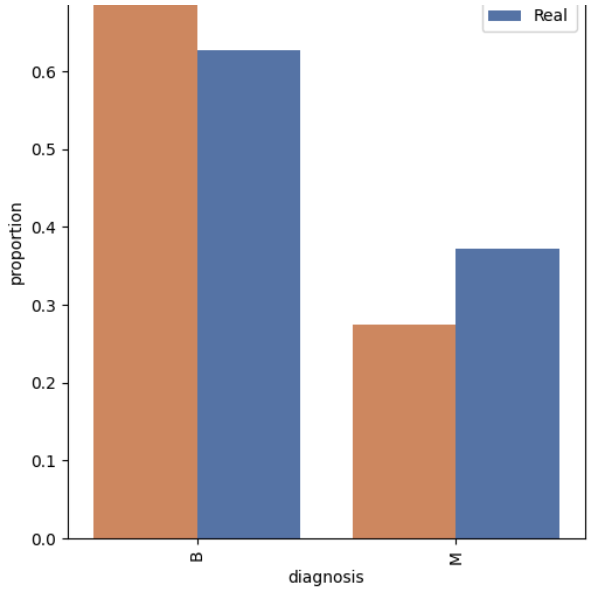

Absolute Log Mean and STDs of numeric data
Means of real and fake data
fake data mean (log)
real data mean (log)
Stds of real and fake data
fake data std (log)
real data std (log)
Cumulative Sums per feature
Real
Fake
Cumsum
diagnosis
radius_mean
texture_mean
area_mean
smoothness_mean
compactness_mean

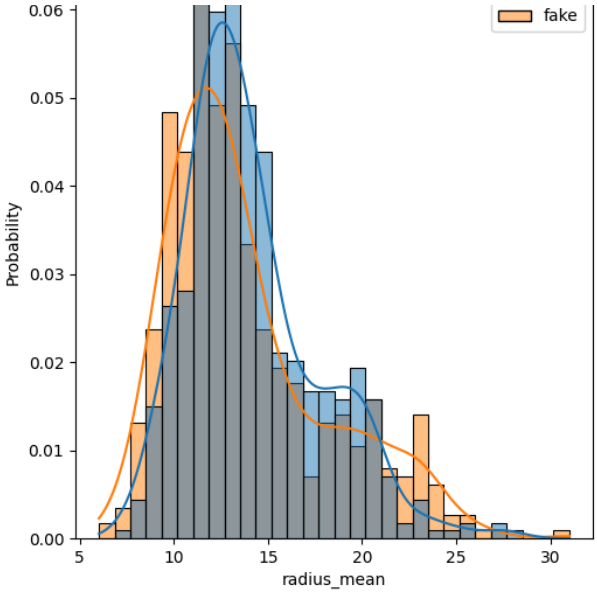

Cumsum
concave points_mean
symmetry_mean
fractal_dimension_mean
texture_se
perimeter_se
area_se
compactness_se
concavity_se
concave points_se
Real
Fake

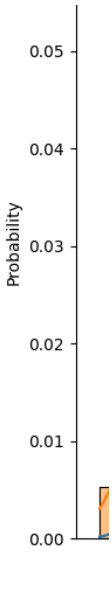


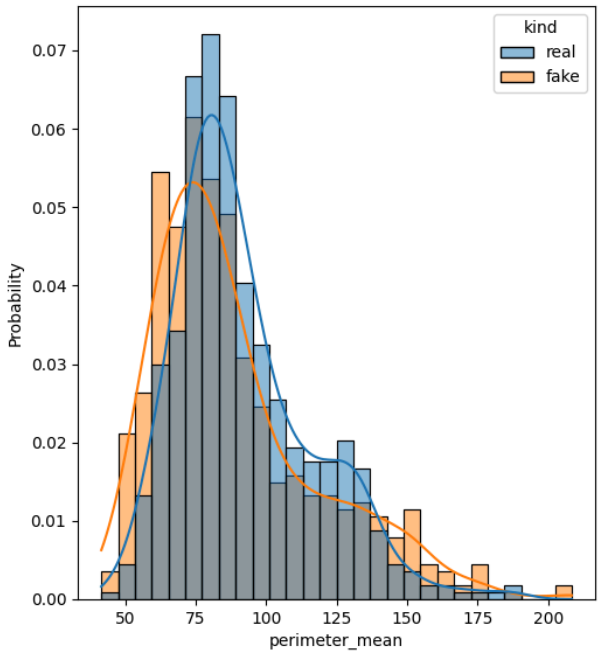

```
/usr/local/lib/python3.10/dist-packages/table_evaluator/table_evaluator.py:182: UserWarning:

FixedFormatter should only be used together with FixedLocator
```

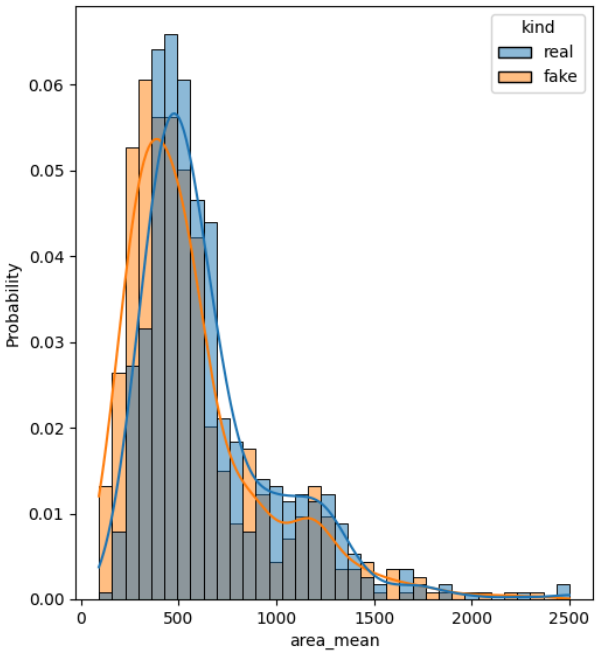


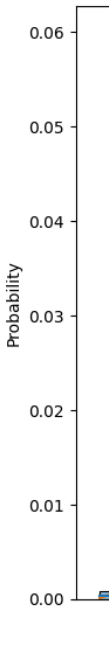

Real
proportion
diagnosis
B
M

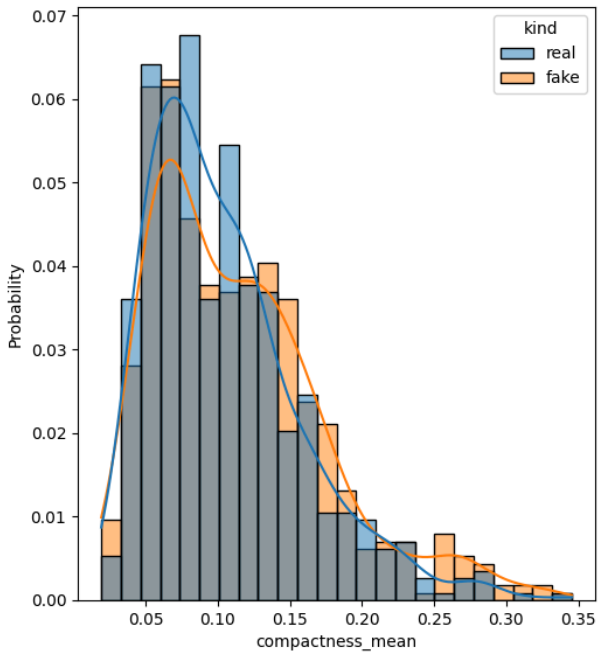

fake
Probability
radius_mean

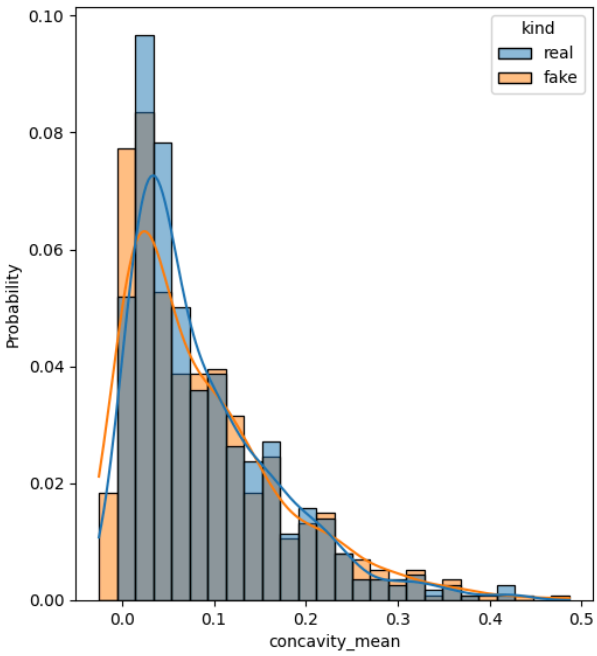

Probability

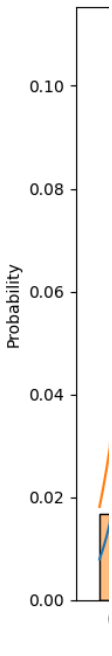

kind
real
fake
Probability
perimeter_mean

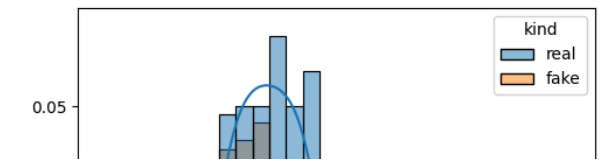

kind
real
fake
Probability
area_mean


Probability

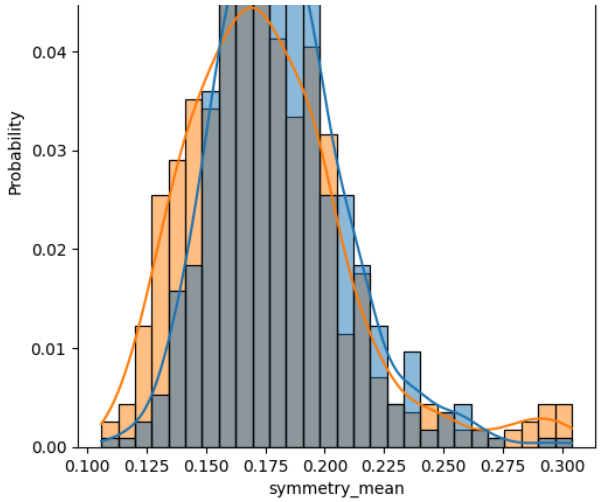

kind
real
fake
Probability
compactness_mean

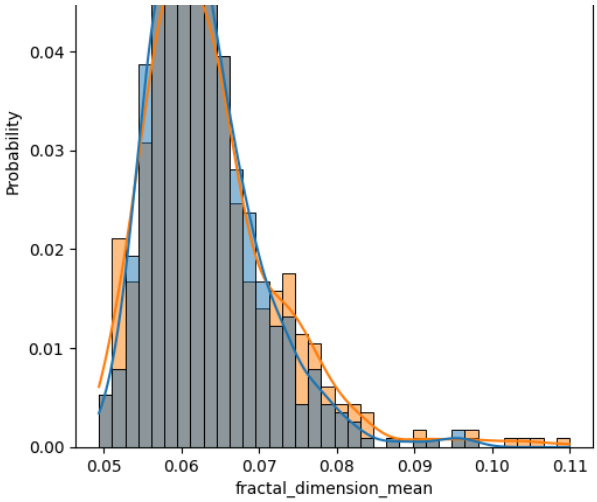

kind
real
fake
Probability
concavity_mean

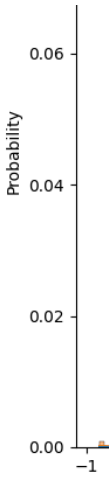

Probability

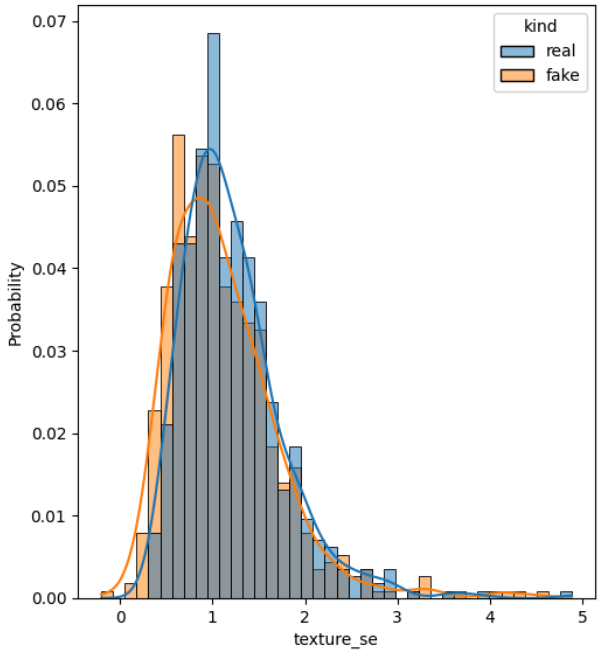

kind
real
fake

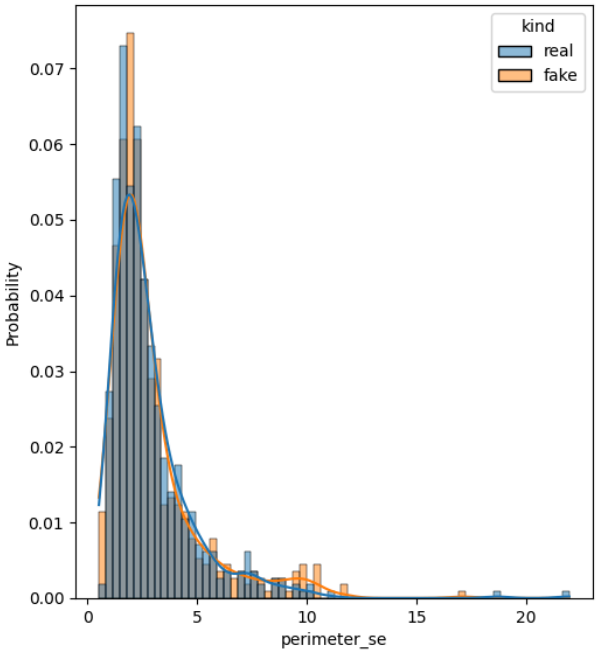

kind
real
fake

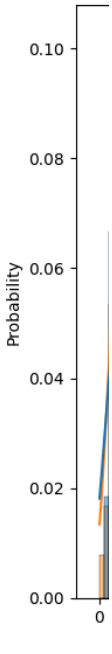

Probability
symmetry_mean

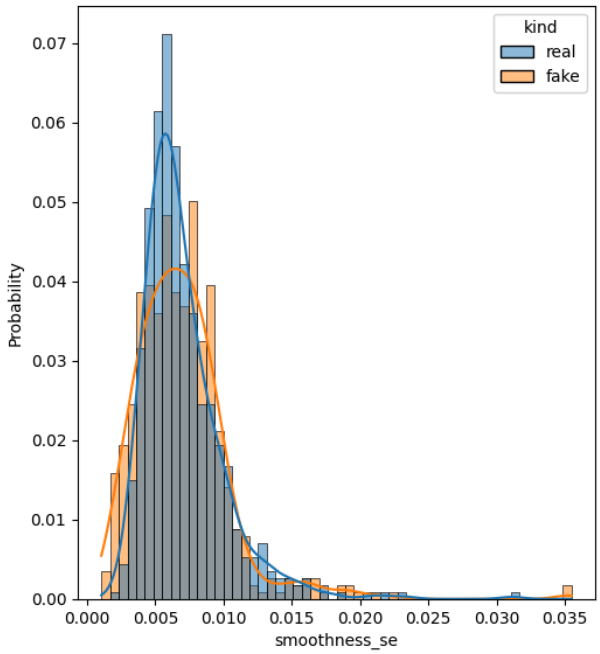

Probability
fractal_dimension_mean

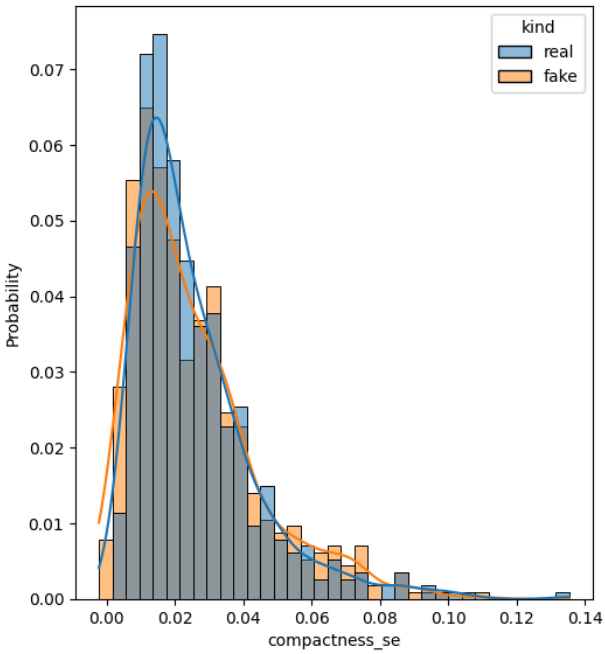

Probability

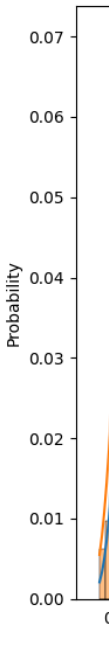

kind
real
fake
Probability
texture_se

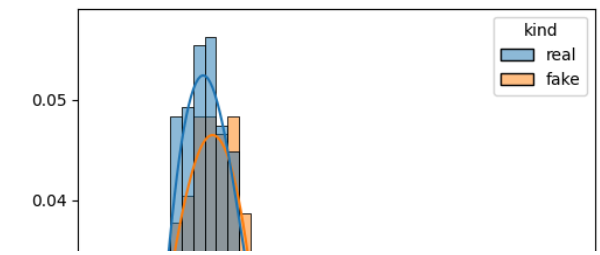

kind
real
fake
Probability
perimeter_se

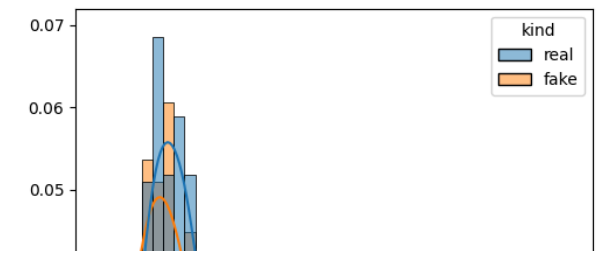

Probability

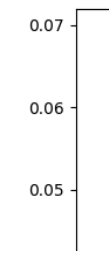

kind
real
fake
Probability
smoothness_se

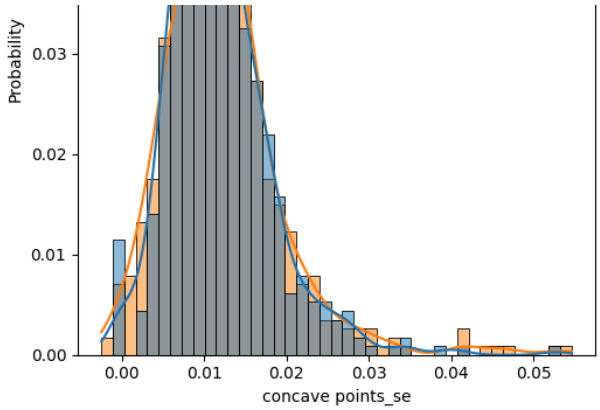

kind
real
fake
Probability
compactness_se

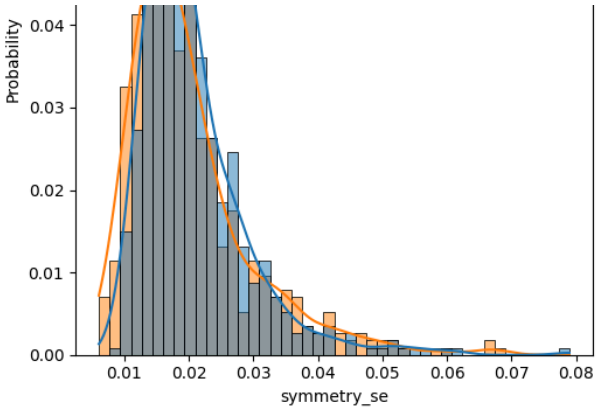

Probability

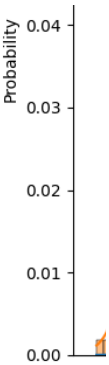

kind
real
fake

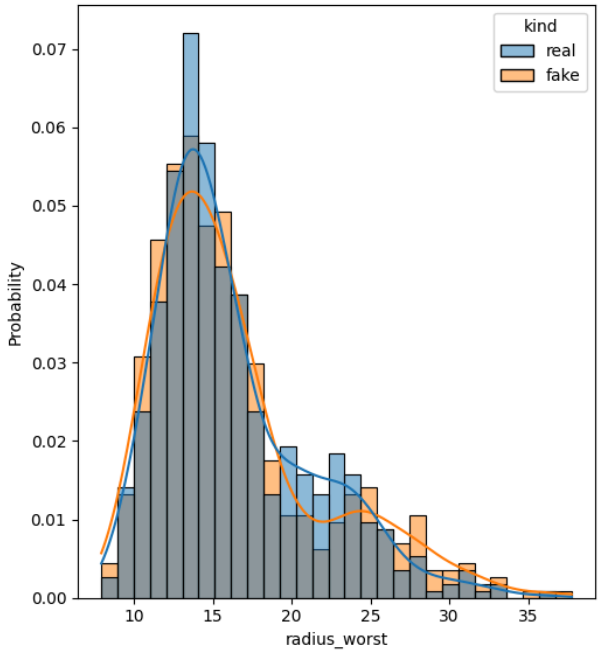

kind
real
fake

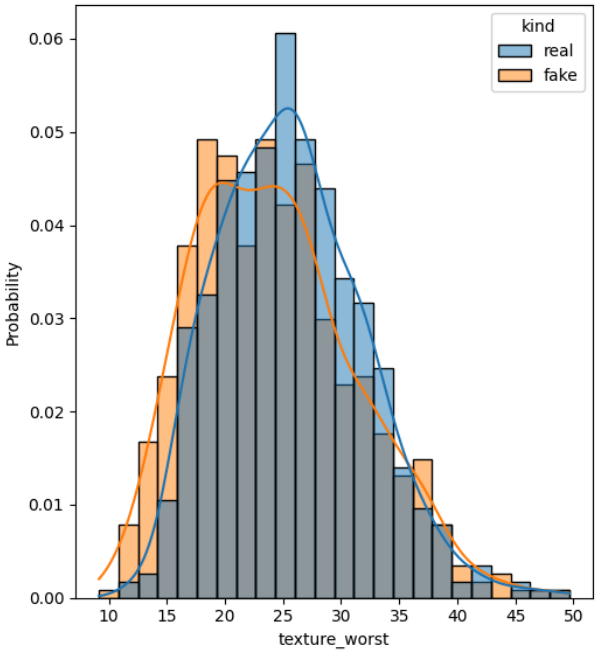


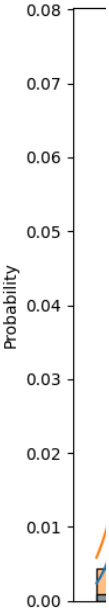


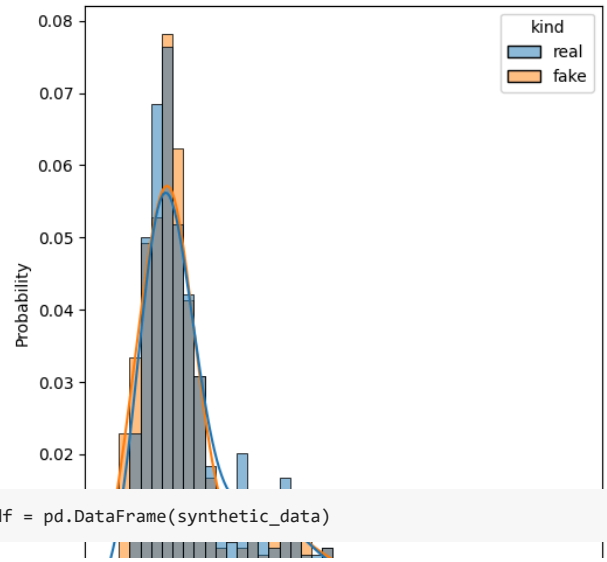


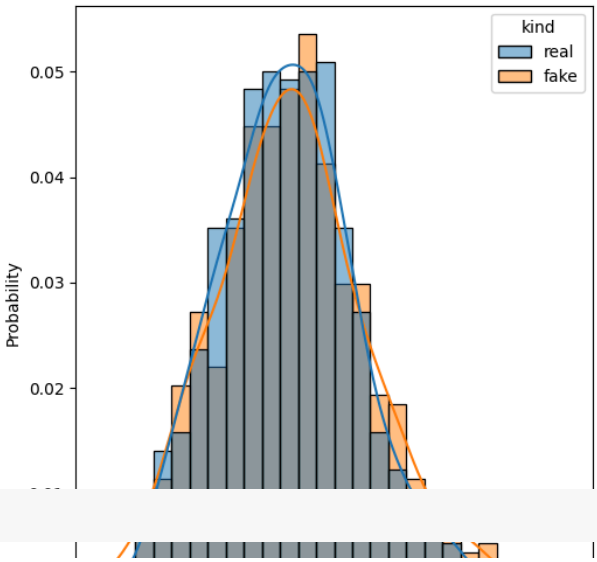


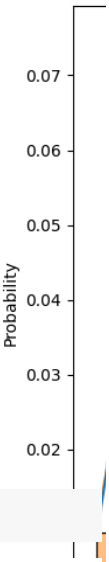













```
syn_df = pd.DataFrame(synthetic_data)
```

LINEAR REGRESSION

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
```

```
df_without_species = transformed_df.drop(columns=['diagnosis'])
df_without_species.dropna()
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.
...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.

```
X_train_old,X_test_old,Y_train_old,Y_test_old = train_test_split(df_without_species,transformed_df['diagnosis'],test_size=0.2,random_st
```

```
new_df = ht.transform(syn_df)
new_df_without_species = new_df.drop(columns=['diagnosis'])
```

```
X_train_new,X_test_new,Y_train_new,Y_test_new = train_test_split(new_df_without_species,new_df['diagnosis'],test_size=0.2,random_state=
```

```
model_old = LinearRegression()
model_old.fit(X_train_old,Y_train_old)
```

```
▾ LinearRegression
LinearRegression()
```

```
# Trained on original data and tested on original data
score_old_old = model_old.score(X_test_old,Y_test_old)
```