In [124…
```
import pandas as pd
import sklearn
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn import datasets
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score, recall_score, precision_score, f
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
import warnings
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report
```

In [125…
```
warnings.filterwarnings('ignore')
# Read in our dataset
cancer_data = pd.read_csv('project1.csv')
```

In [125…
```
# Identify the shape of raw data
cancer_data.shape
```

Out[125… (386, 10)

In [125…
```
# Identify the sie of raw data
cancer_data.size
```

Out[125… 3860

In [125…
```
# Get info about datatype
cancer_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 386 entries, 0 to 385
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   class        386 non-null    object
 1   age          386 non-null    object
 2   menopause    386 non-null    object
 3   tumor-size   385 non-null    object
 4   inv-nodes    385 non-null    object
 5   node-caps    386 non-null    object
 6   deg-malig    386 non-null    int64 
 7   breast       386 non-null    object
 8   breast-quad  386 non-null    object
 9   irradiat     386 non-null    object
dtypes: int64(1), object(9)
memory usage: 30.3+ KB
```

In [125…
```
# cast column vals to type category
cancer_data['class'] = cancer_data['class'].astype("category")
cancer_data['age'] = cancer_data['age'].astype("category")
cancer_data['menopause'] = cancer_data['menopause'].astype("category")
cancer_data['tumor-size'] = cancer_data['tumor-size'].astype("category")
```

```
cancer_data['inv-nodes'] = cancer_data['inv-nodes'].astype("category")
cancer_data['node-caps'] = cancer_data['node-caps'].astype("category")
cancer_data['breast'] = cancer_data['breast'].astype("category")
cancer_data['breast-quad'] = cancer_data['breast-quad'].astype("category")
cancer_data['irradiat'] = cancer_data['irradiat'].astype("category")
```

In [125…
```
# Check if any duplicate rows exist in the data
cancer_data.duplicated().sum()
```

Out[125… 11

In [125…
```
# Treat duplicated rows
cancer_data = cancer_data.drop_duplicates()
```

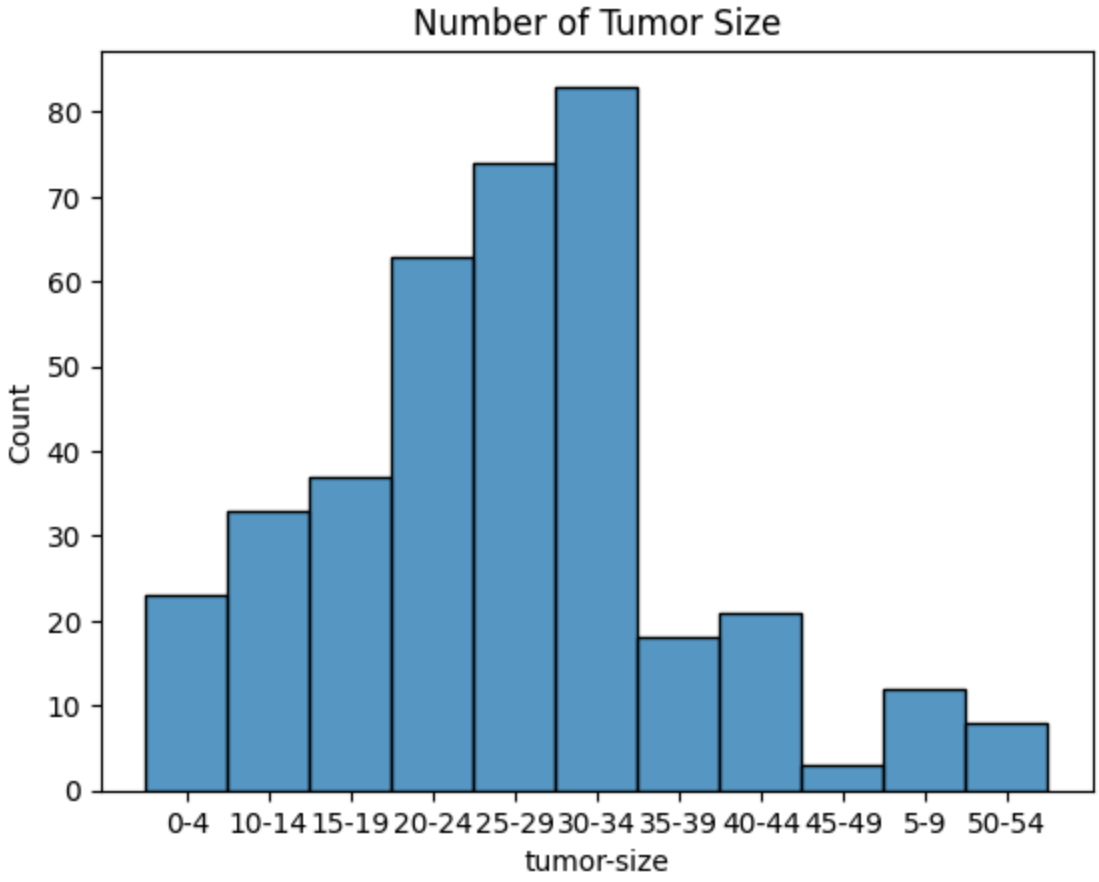
In [125…
```
# Check to see if any missing data exists in the dataset
cancer_data.isnull().sum()
```

Out[125…
```
class          0
age            0
menopause      0
tumor-size     1
inv-nodes      1
node-caps      0
deg-malig      0
breast         0
breast-quad    0
irradiat       0
dtype: int64
```

In [125…
```
# Treating missing values from 'tumor-size' and 'inv-nodes' columns with the
cancer_data['tumor-size'] = cancer_data.groupby(['breast-quad'])['tumor-size
cancer_data['inv-nodes'] = cancer_data.groupby(['breast-quad'])['inv-nodes']
```

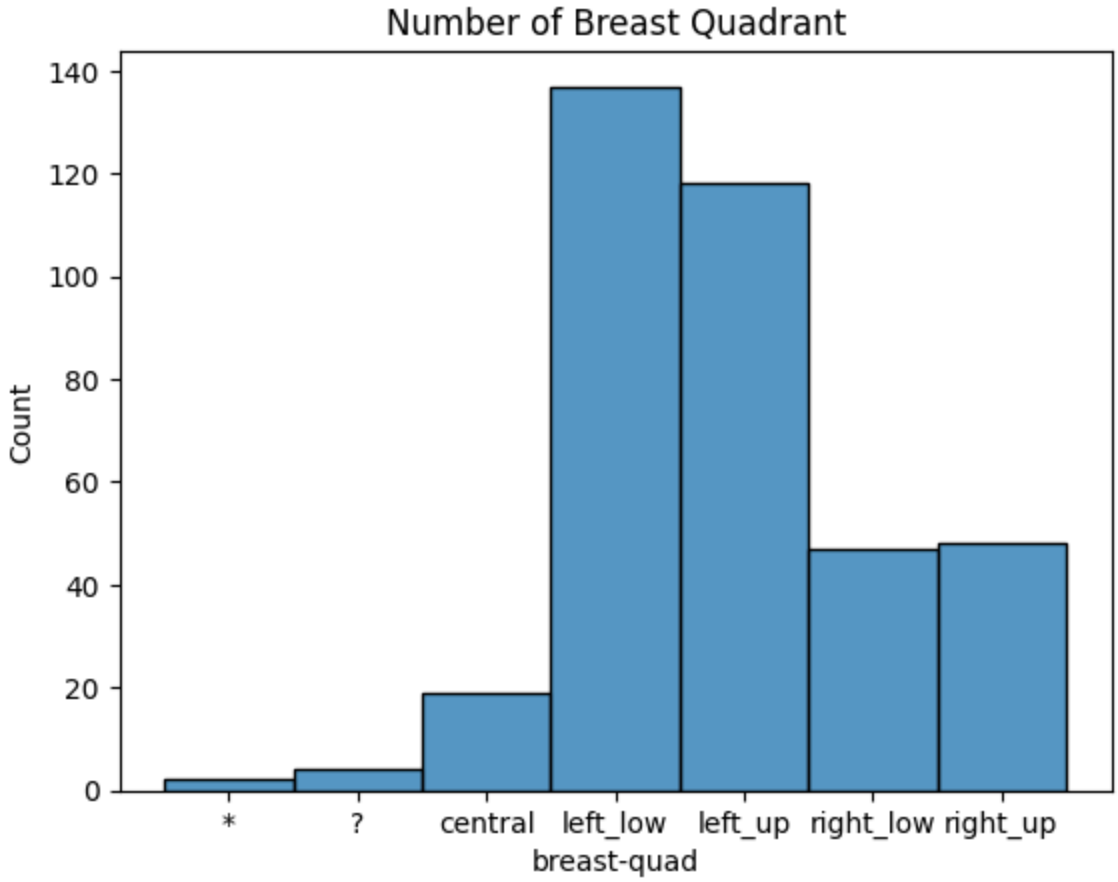
In [125…
```
# Histogram Counting the tumor size
sns.histplot(data=cancer_data['tumor-size'])
plt.title("Number of Tumor Size")
```

Out[125… Text(0.5, 1.0, 'Number of Tumor Size')



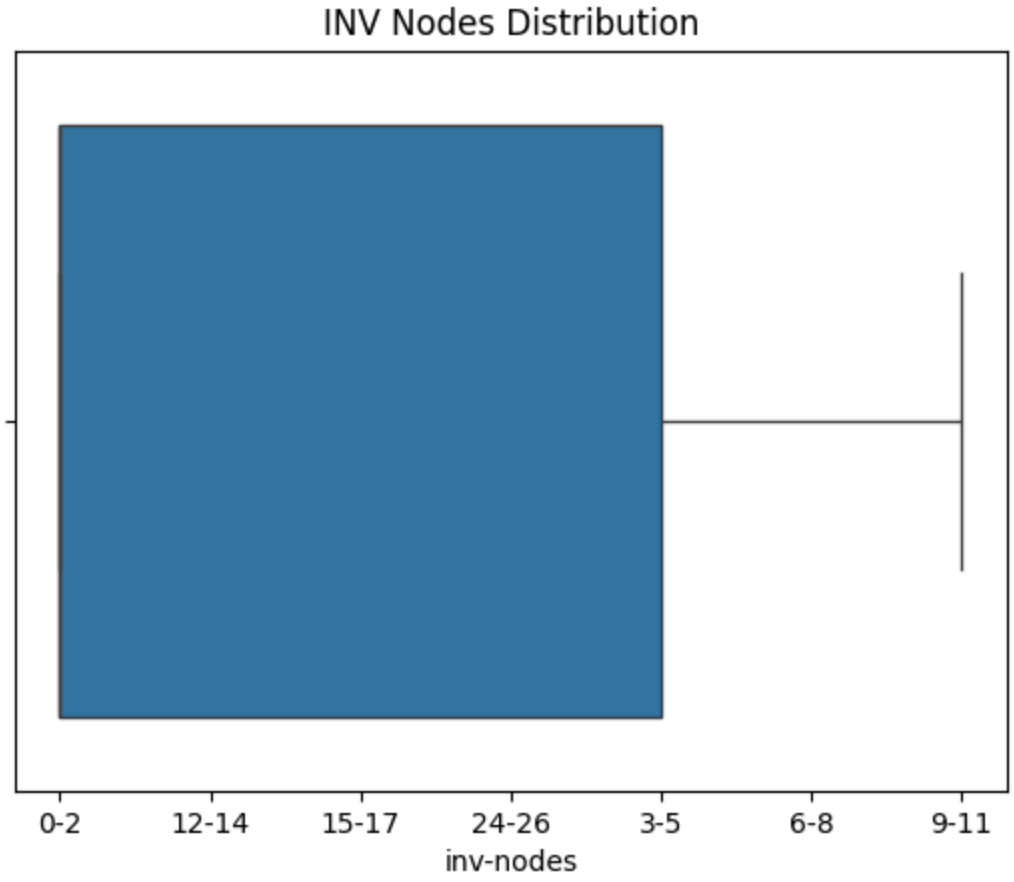
In [126…
```
# Histogram Counting the location of the tumor
sns.histplot(data=cancer_data['breast-quad'])
plt.title("Number of Breast Quadrant")
```

Out[126…
```
Text(0.5, 1.0, 'Number of Breast Quadrant')
```



In [126…
```
# Box Plot for Inv-Nodes Column
sns.boxplot(data=cancer_data, x="inv-nodes")
plt.title("INV Nodes Distribution")
```

Out[126… `Text(0.5, 1.0, 'INV Nodes Distribution')`



In [126…

```
# Performing One hot encoding on Categorical Columns
cancer_data = pd.get_dummies(cancer_data, columns=["class", "menopause", "ag
cancer_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 375 entries, 0 to 385
Data columns (total 36 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   deg-malig                375 non-null    int64
 1   class_recurrence-events  375 non-null    bool
 2   menopause_lt40           375 non-null    bool
 3   menopause_premeno        375 non-null    bool
 4   age_30-39                375 non-null    bool
 5   age_40-49                375 non-null    bool
 6   age_50-59                375 non-null    bool
 7   age_60-69                375 non-null    bool
 8   age_70-79                375 non-null    bool
 9   tumor-size_10-14         375 non-null    bool
 10  tumor-size_15-19         375 non-null    bool
 11  tumor-size_20-24         375 non-null    bool
 12  tumor-size_25-29         375 non-null    bool
 13  tumor-size_30-34         375 non-null    bool
 14  tumor-size_35-39         375 non-null    bool
 15  tumor-size_40-44         375 non-null    bool
 16  tumor-size_45-49         375 non-null    bool
 17  tumor-size_5-9           375 non-null    bool
 18  tumor-size_50-54         375 non-null    bool
 19  inv-nodes_12-14          375 non-null    bool
 20  inv-nodes_15-17          375 non-null    bool
 21  inv-nodes_24-26          375 non-null    bool
 22  inv-nodes_3-5            375 non-null    bool
 23  inv-nodes_6-8            375 non-null    bool
 24  inv-nodes_9-11           375 non-null    bool
 25  node-caps_?              375 non-null    bool
 26  node-caps_no             375 non-null    bool
 27  node-caps_yes            375 non-null    bool
 28  breast_right             375 non-null    bool
 29  breast-quad_?            375 non-null    bool
 30  breast-quad_central      375 non-null    bool
 31  breast-quad_left_low     375 non-null    bool
 32  breast-quad_left_up      375 non-null    bool
 33  breast-quad_right_low    375 non-null    bool
 34  breast-quad_right_up     375 non-null    bool
 35  irradiat_yes             375 non-null    bool
dtypes: bool(35), int64(1)
memory usage: 18.7 KB
```

In [126…

```
# Assigning subset of data as X and y
X = cancer_data.drop('class_recurrence-events', axis=1)
y = cancer_data["class_recurrence-events"]
```

In [126…

```
# Linear Classification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, r
clf = SGDClassifier(loss="perceptron", alpha=0.1)
clf.fit(X_train, y_train)
```

Out[126…

SGDClassifier

```
SGDClassifier(alpha=0.1, loss='perceptron')
```

In [126…

```
# Accuracy for Linear Classification
accuracy_test = accuracy_score(y_test, clf.predict(X_test))
accuracy_train = accuracy_score(y_train, clf.predict(X_train))

print(f"Test Accuracy: {accuracy_test:.4f}")
print(f"Train Accuracy: {accuracy_train:.4f}\n")

# Recall for Linear Classification
recall_test = recall_score(y_test, clf.predict(X_test), average='weighted')
recall_train = recall_score(y_train, clf.predict(X_train), average='weighted

print(f"Test Recall: {recall_test:.4f}")
print(f"Train Recall: {recall_train:.4f}\n")

# Precision for Linear Classification
precision_test = precision_score(y_test, clf.predict(X_test), average='weigh
precision_train = precision_score(y_train, clf.predict(X_train), average='we

print(f"Test Precision: {precision_test:.4f}")
print(f"Train Precision: {precision_train:.4f}\n")

# F-1 Score for Linear Classification
f1_test = f1_score(y_test, clf.predict(X_test), average='weighted')
f1_train = f1_score(y_train, clf.predict(X_train), average='weighted')

print(f"Test F1 Score: {f1_test:.4f}")
print(f"Train F1 Score: {f1_train:.4f}\n")
print(classification_report(y_test, clf.predict(X_test)))
```

```
Test Accuracy: 0.5398
Train Accuracy: 0.6031

Test Recall: 0.5398
Train Recall: 0.6031

Test Precision: 0.6558
Train Precision: 0.7092

Test F1 Score: 0.5526
Train F1 Score: 0.6112

              precision    recall  f1-score   support

       False       0.78      0.46      0.58        78
        True       0.37      0.71      0.49        35

    accuracy                           0.54       113
   macro avg       0.58      0.59      0.54       113
weighted avg       0.66      0.54      0.55       113
```

In [126…
```
# Instantiate the model for K-Nearest Neighbor Classifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
```

Out[126…
```
KNeighborsClassifier(n_neighbors=3)
```

In [126…
```
# Accuracy for K-Nearest Neighbor Classifier
accuracy_test = accuracy_score(y_test, knn.predict(X_test))
accuracy_train = accuracy_score(y_train, knn.predict(X_train))

print(f"Test Accuracy: {accuracy_test:.4f}")
print(f"Train Accuracy: {accuracy_train:.4f}\n")

# Recall for K-Nearest Neighbor Classifier
recall_test = recall_score(y_test, knn.predict(X_test), average='weighted')
recall_train = recall_score(y_train, knn.predict(X_train), average='weighted

print(f"Test Recall: {recall_test:.4f}")
print(f"Train Recall: {recall_train:.4f}\n")

# Precision for K-Nearest Neighbor Classifier
precision_test = precision_score(y_test, knn.predict(X_test), average='weigh
precision_train = precision_score(y_train, knn.predict(X_train), average='we

print(f"Test Precision: {precision_test:.4f}")
print(f"Train Precision: {precision_train:.4f}\n")

# F-1 Score for K-Nearest Neighbor Classifier
f1_test = f1_score(y_test, knn.predict(X_test), average='weighted')
f1_train = f1_score(y_train, knn.predict(X_train), average='weighted')

print(f"Test F1 Score: {f1_test:.4f}")
print(f"Train F1 Score: {f1_train:.4f}\n")

print(classification_report(y_test, knn.predict(X_test)))
```

```
Test Accuracy: 0.6460
Train Accuracy: 0.7672

Test Recall: 0.6460
Train Recall: 0.7672

Test Precision: 0.5988
Train Precision: 0.7592

Test F1 Score: 0.6100
Train F1 Score: 0.7575

              precision    recall  f1-score   support

       False       0.70      0.85      0.77        78
        True       0.37      0.20      0.26        35

    accuracy                           0.65       113
   macro avg       0.54      0.52      0.51       113
weighted avg       0.60      0.65      0.61       113
```

In [126…
```
knn_search_accuracy = KNeighborsClassifier()
param_grid = {"n_neighbors": np.arange(1, 100)}
knn_gscv = GridSearchCV(knn_search_accuracy, param_grid, cv=5)
knn_gscv.fit(X_train, y_train)
best_knn = knn_gscv.best_estimator_
```

In [126…
```
knn_search_recall = KNeighborsClassifier()
knn_gscv2 = GridSearchCV(knn_search_recall, param_grid, cv=5, scoring="recal
knn_gscv2.fit(X_train, y_train)
best_recall = knn_gscv2.best_estimator_
```

In [127…
```
# Recall for K-Nearest Neighbor Classifier using Grid search CV
models = [knn, best_knn, best_recall]
for m in models:

    recall_test=recall_score(y_test, m.predict(X_test), average='weighted')
    print(f'Recall of {m} on test data is : {recall_test}')
    recall_train=recall_score(y_train, m.predict(X_train), average='weighted
    print(f'Recall of {m} on train data is : {recall_train}\n')
```

```
Recall of KNeighborsClassifier(n_neighbors=3) on test data is : 0.6460176991
150443
Recall of KNeighborsClassifier(n_neighbors=3) on train data is : 0.767175572
519084

Recall of KNeighborsClassifier(n_neighbors=12) on test data is : 0.672566371
681416
Recall of KNeighborsClassifier(n_neighbors=12) on train data is : 0.68702290
07633588

Recall of KNeighborsClassifier(n_neighbors=1) on test data is : 0.5486725663
716814
Recall of KNeighborsClassifier(n_neighbors=1) on train data is : 0.988549618
3206107
```

In [127…

```python
# Accuracy for K-Nearest Neighbor Classifier using Grid search CV
accuracy_test = accuracy_score(y_test, best_knn.predict(X_test))
accuracy_train = accuracy_score(y_train, best_knn.predict(X_train))

print(f"Test Accuracy: {accuracy_test:.4f}")
print(f"Train Accuracy: {accuracy_train:.4f}\n")

# Precision for K-Nearest Neighbor Classifier using Grid search CV
precision_test = precision_score(y_test, best_knn.predict(X_test), average='
precision_train = precision_score(y_train, best_knn.predict(X_train), averag

print(f"Test Precision: {precision_test:.4f}")
print(f"Train Precision: {precision_train:.4f}\n")

# F-1 Score for K-Nearest Neighbor Classifier using Grid search CV
f1_test = f1_score(y_test, best_knn.predict(X_test), average='weighted')
f1_train = f1_score(y_train, best_knn.predict(X_train), average='weighted')

print(f"Test F1 Score: {f1_test:.4f}")
print(f"Train F1 Score: {f1_train:.4f}\n")

print(classification_report(y_test, best_knn.predict(X_test)))
```

```
Test Accuracy: 0.6726
Train Accuracy: 0.6870

Test Precision: 0.5524
Train Precision: 0.6600

Test F1 Score: 0.5696
Train F1 Score: 0.6223
```

	precision	recall	f1-score	support
False	0.69	0.96	0.80	78
True	0.25	0.03	0.05	35
accuracy			0.67	113
macro avg	0.47	0.50	0.43	113
weighted avg	0.55	0.67	0.57	113