in this code we will apply the fundamentals of KNN and Classificaion metrics

this data about medical field (Breast Cancer) and during preprocessing we will show it

Libraries

In [199…

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns
```

In [200…

```
df=pd.read_csv('data.csv')
```

In [201…

```
df .head()
```

Out[201…

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothn
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	

5 rows × 33 columns

In [202…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 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
 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
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

In [203…

```
print(f"The number of nulls is {df.isna().sum()} \n")
print(f"The number of duplicates is {df.duplicated().sum()}")
```

```
The number of nulls is id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
Unnamed: 32              569
dtype: int64

The number of duplicates is 0
```

In [204…

```
df=df.drop('Unnamed: 32',axis=1)
```

In [205…

```
# the diagnosis colunms we can convert it to [0,1]
# M=1 & B=0
df['diagnosis']=df['diagnosis'].map({'M':1,'B':0})
```

In [206…

```
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    int64  
 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
 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(2)
memory usage: 142.4 KB
```

beacause the assumtion of KNN we must to check the outliers , do scaling ,check multi coleniarity ,etc...

In [207…

```
df.columns
```

Out[207…

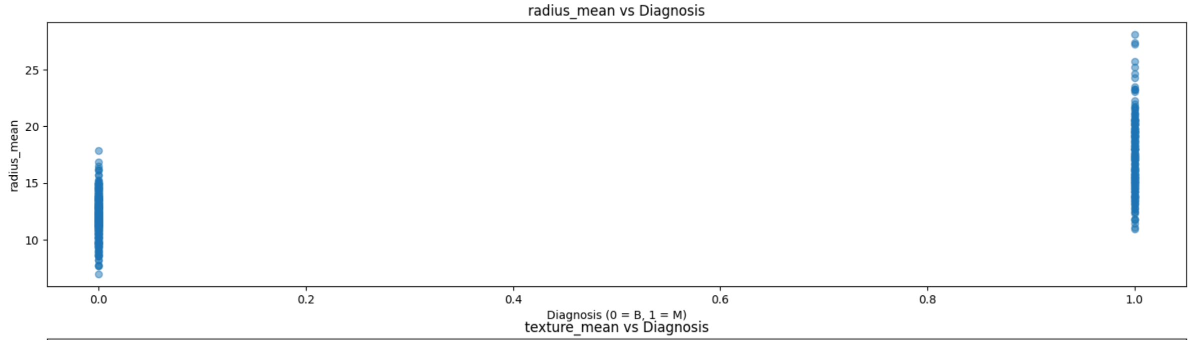
```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

In [208…

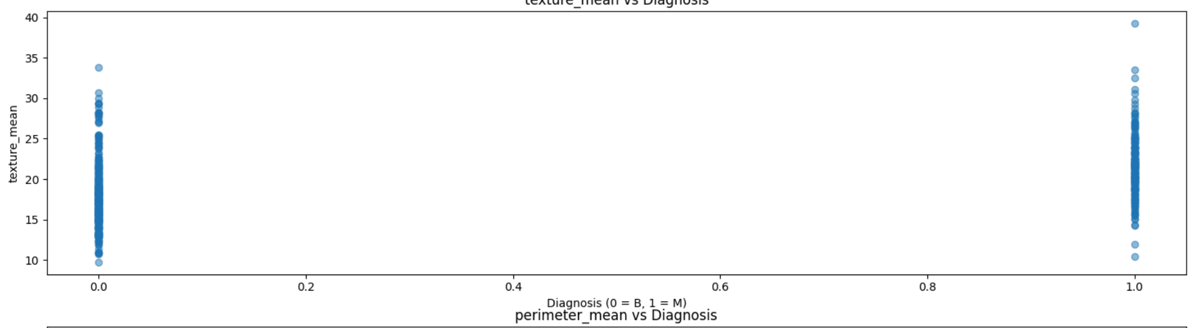
```
features=df.drop(['id','diagnosis'],axis=1).columns
lenght=len(features)
fig , axes=plt.subplots(33, 1, figsize=(18, 33 * 5))
for i,col in enumerate(features):
    axes[i].scatter(df['diagnosis'], df[col], alpha=0.5)
    axes[i].set_title(f'{col} vs Diagnosis')
    axes[i].set_xlabel('Diagnosis (0 = B, 1 = M)')
    axes[i].set_ylabel(col)

plt.show()
```

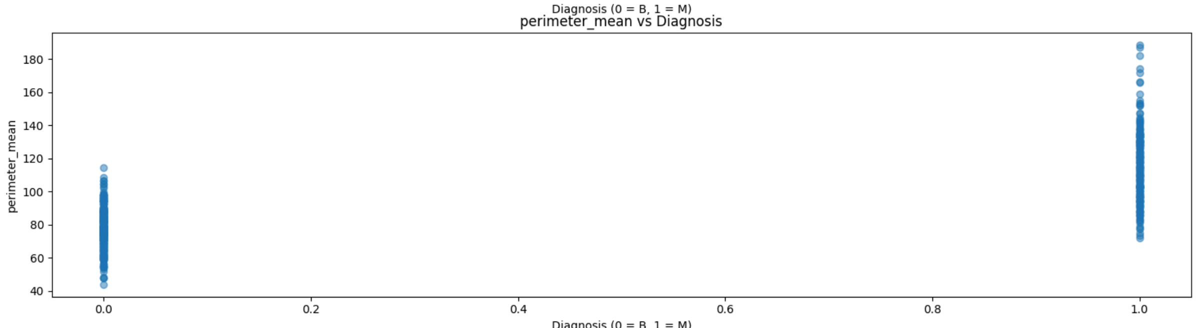
radius_mean vs Diagnosis



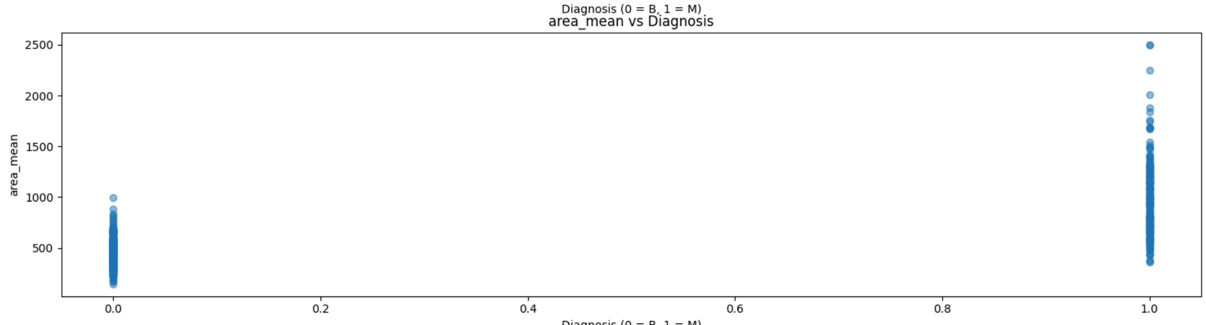
texture_mean vs Diagnosis



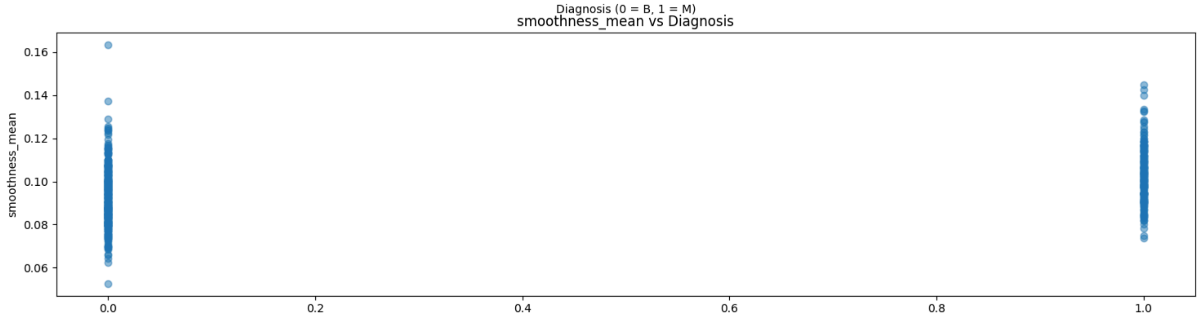
perimeter_mean vs Diagnosis



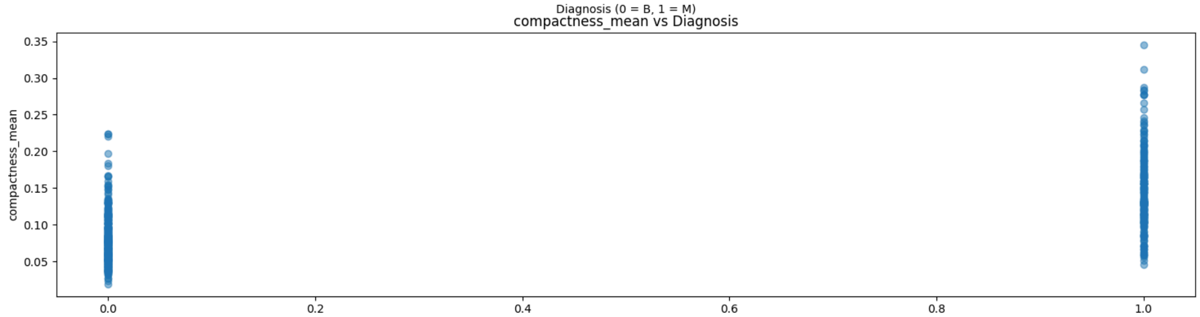
area_mean vs Diagnosis



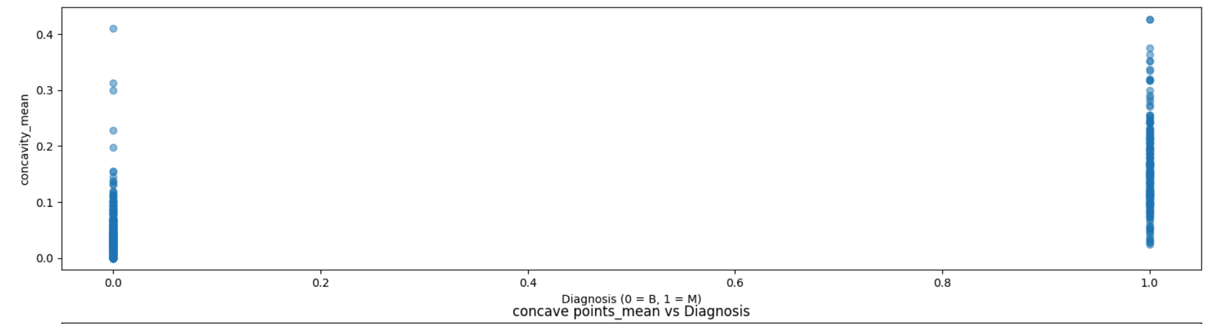
smoothness_mean vs Diagnosis



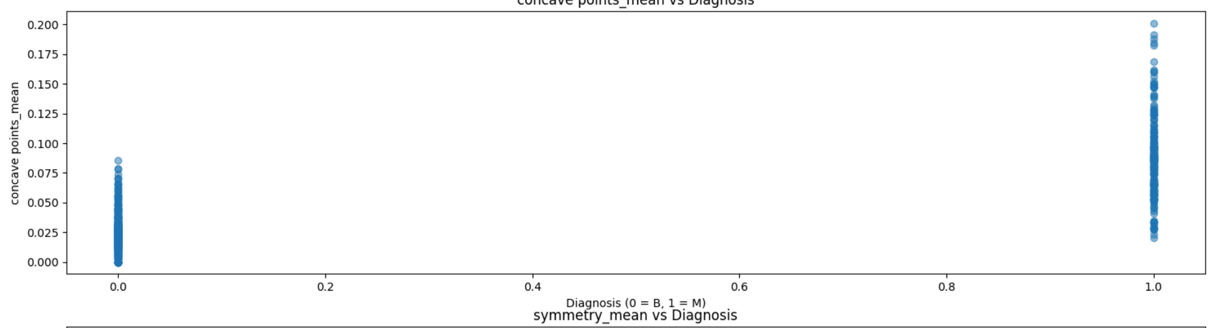
compactness_mean vs Diagnosis



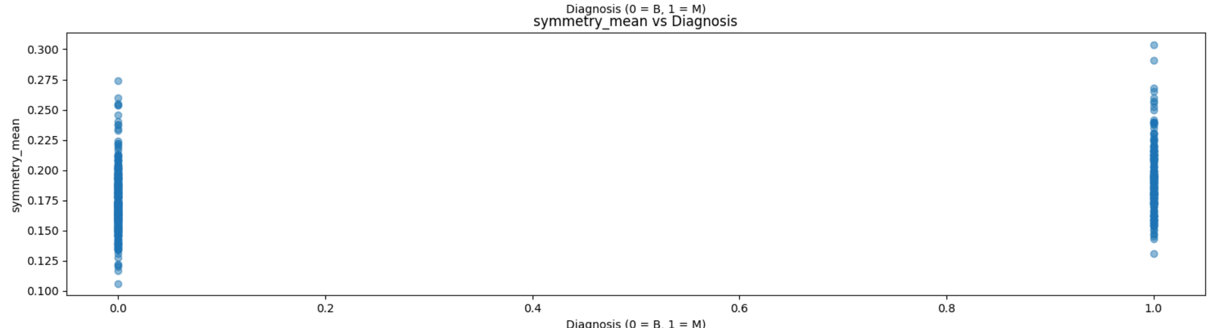
concavity_mean vs Diagnosis



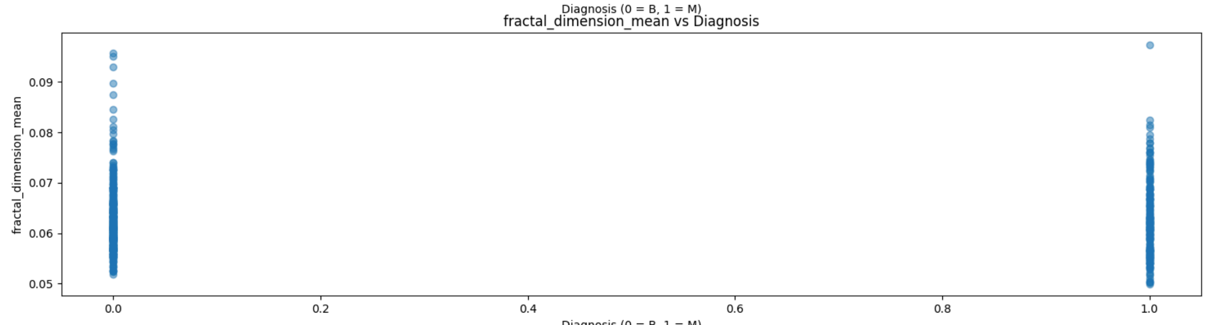
concave points_mean vs Diagnosis



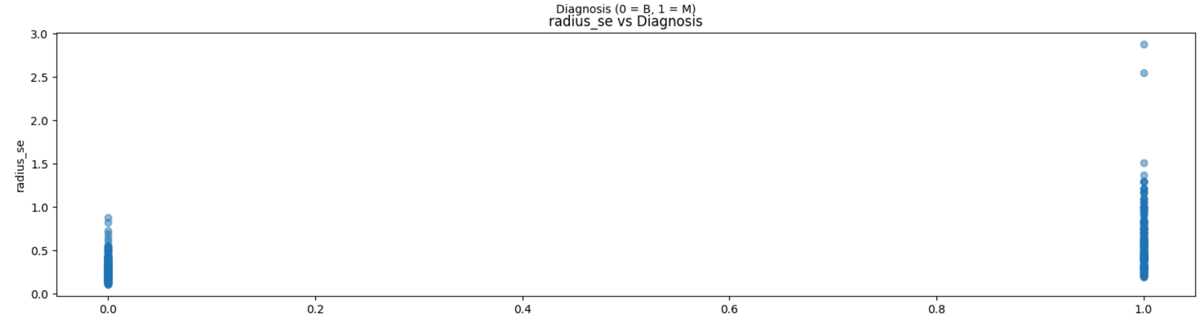
symmetry_mean vs Diagnosis



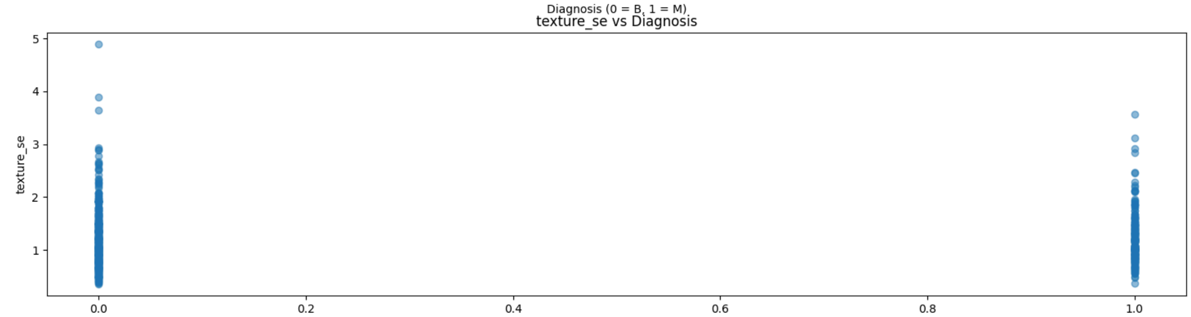
fractal_dimension_mean vs Diagnosis



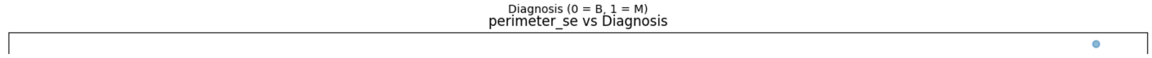
radius_se vs Diagnosis

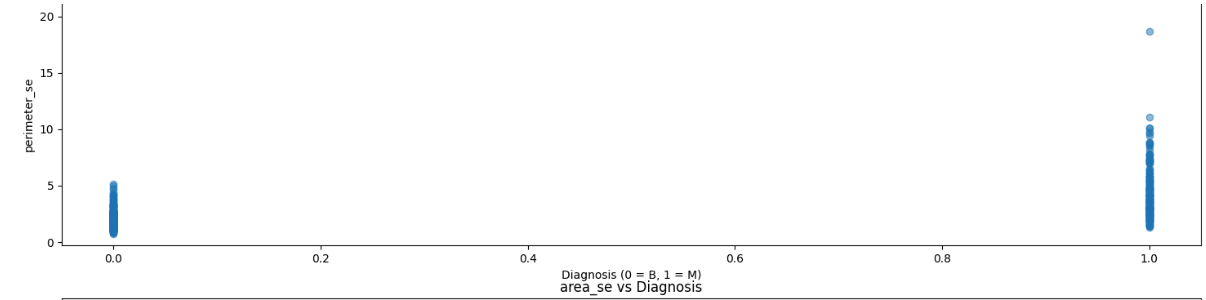


texture_se vs Diagnosis

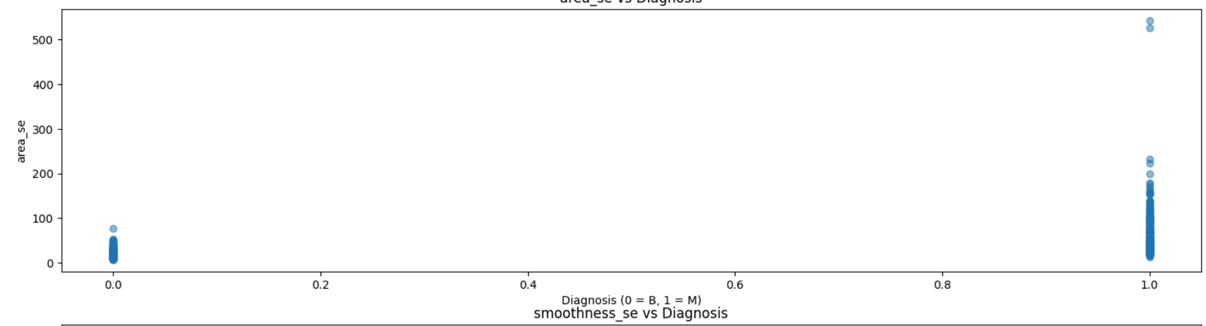


perimeter_se vs Diagnosis

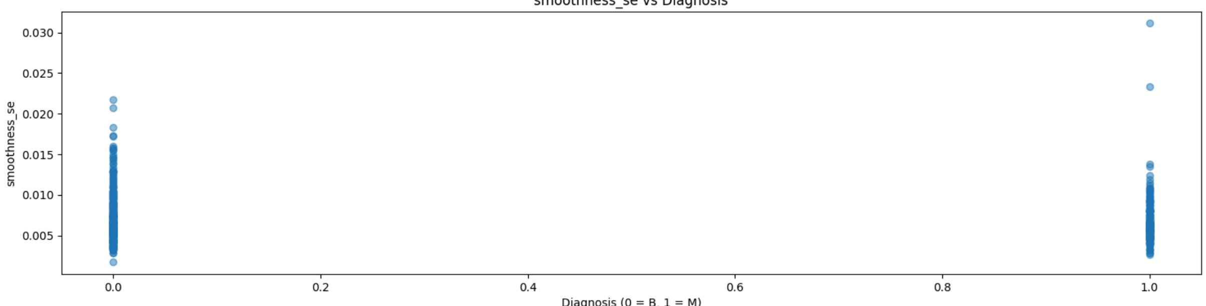




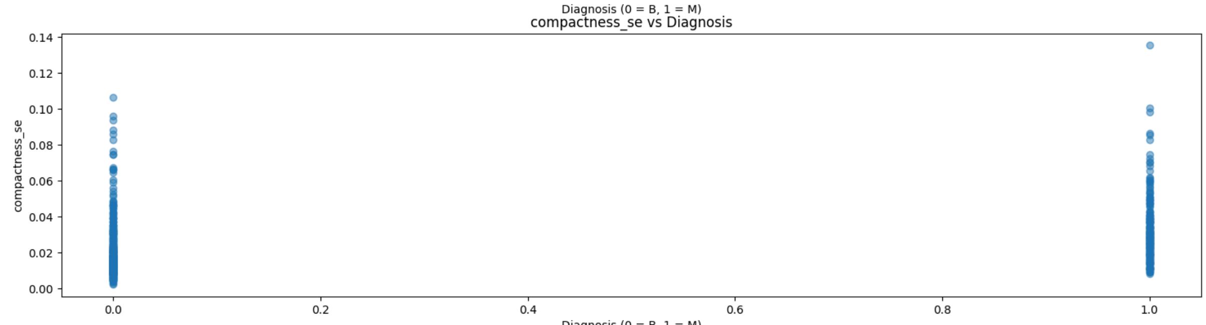
area_se vs Diagnosis



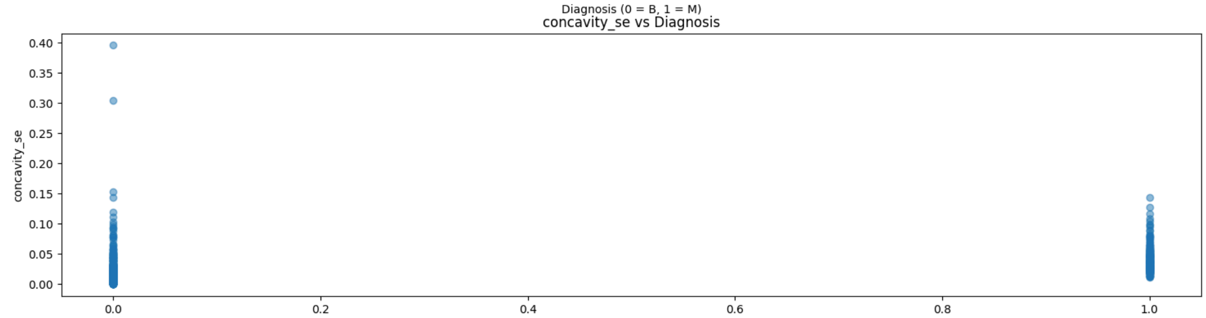
smoothness_se vs Diagnosis



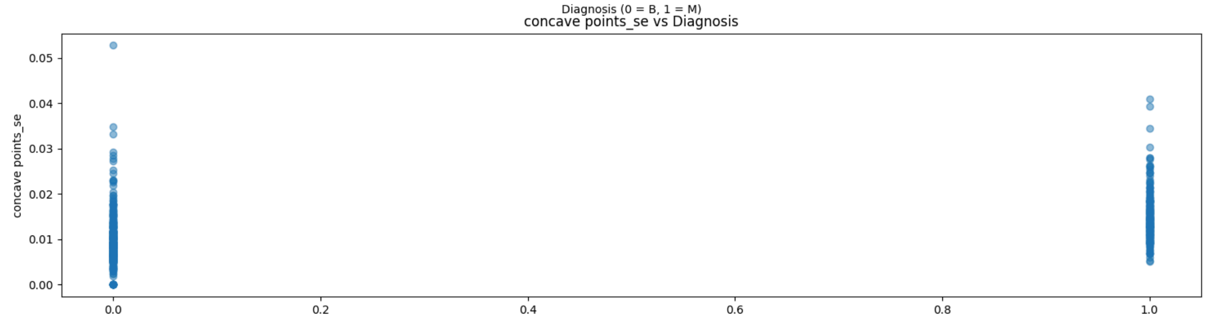
compactness_se vs Diagnosis



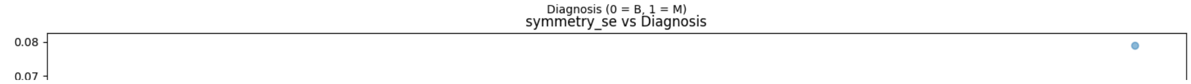
concavity_se vs Diagnosis

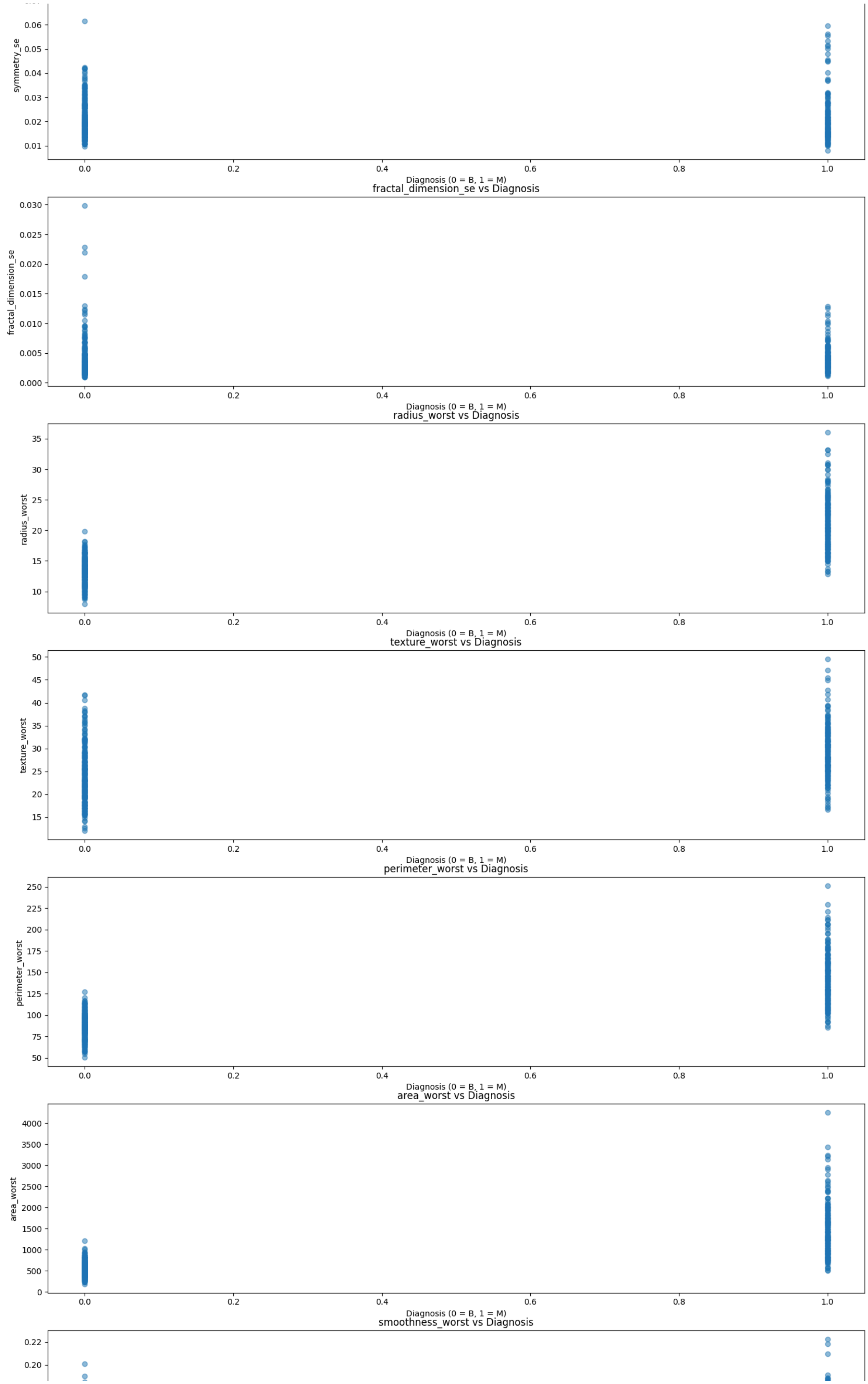


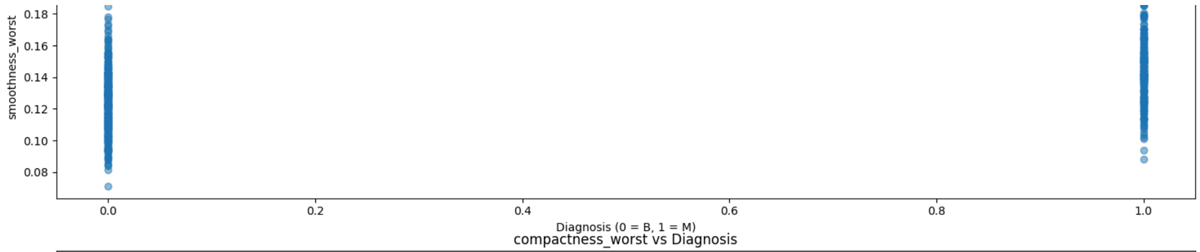
concave points_se vs Diagnosis



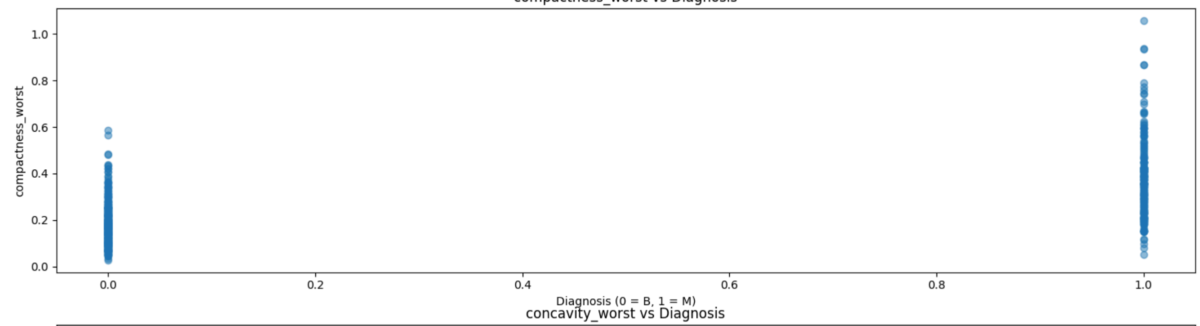
symmetry_se vs Diagnosis



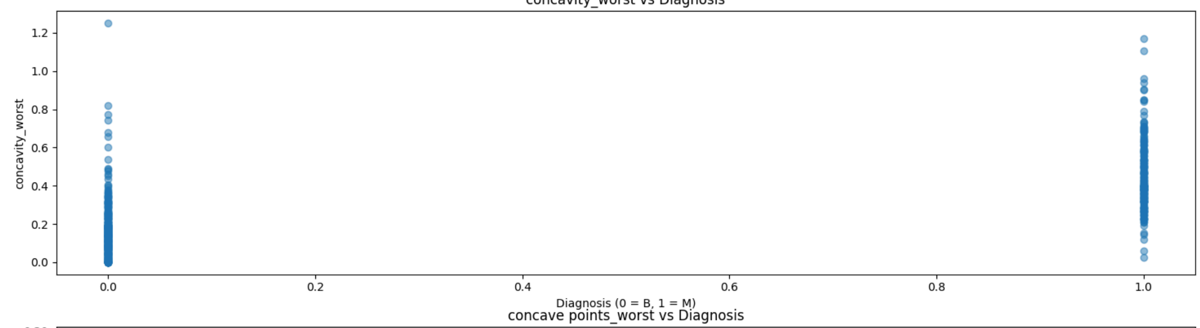

symmetry_se
Diagnosis (0 = B, 1 = M)
fractal_dimension_se vs Diagnosis
fractal_dimension_se
Diagnosis (0 = B, 1 = M)
radius_worst vs Diagnosis
radius_worst
Diagnosis (0 = B, 1 = M)
texture_worst vs Diagnosis
texture_worst
Diagnosis (0 = B, 1 = M)
perimeter_worst vs Diagnosis
perimeter_worst
Diagnosis (0 = B, 1 = M)
area_worst vs Diagnosis
area_worst
Diagnosis (0 = B, 1 = M)
smoothness_worst vs Diagnosis



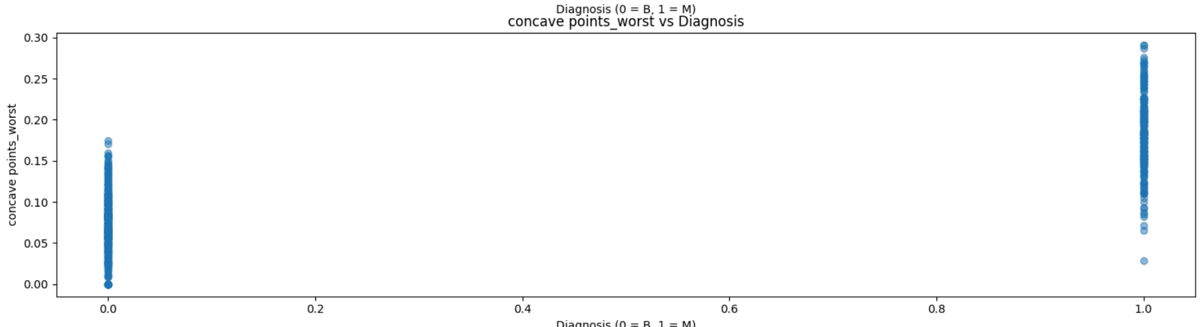
compactness_worst vs Diagnosis



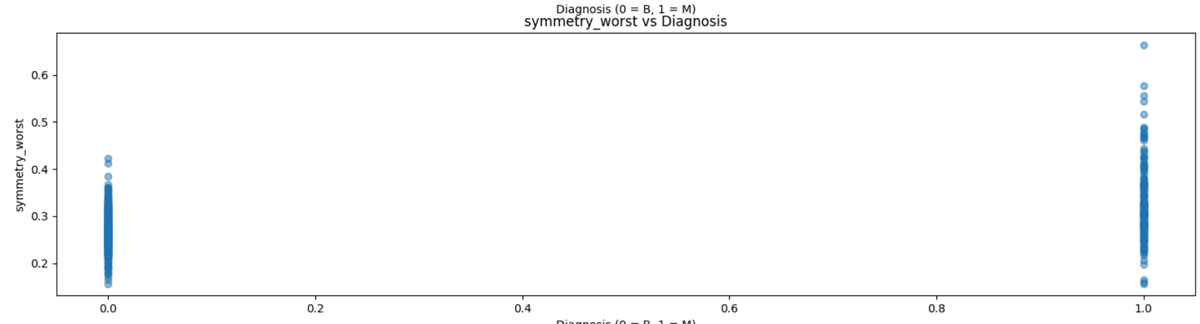
concavity_worst vs Diagnosis



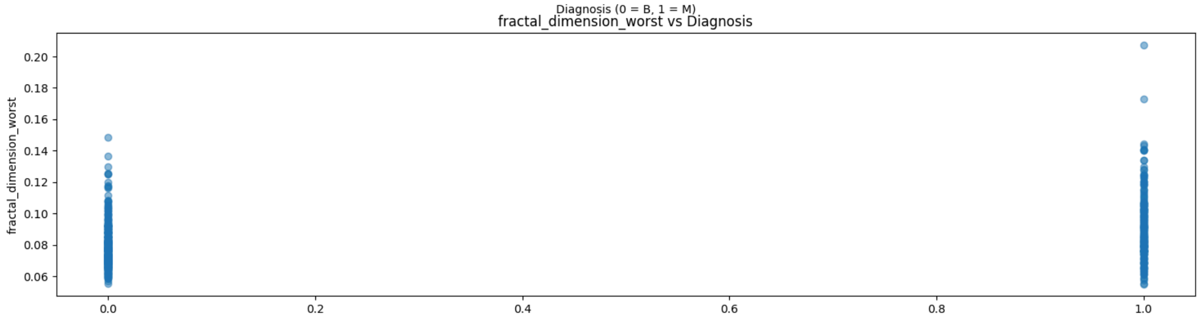
concave points_worst vs Diagnosis

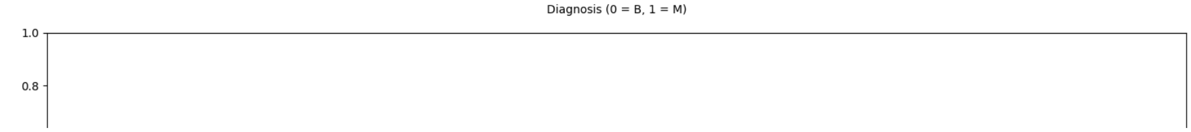


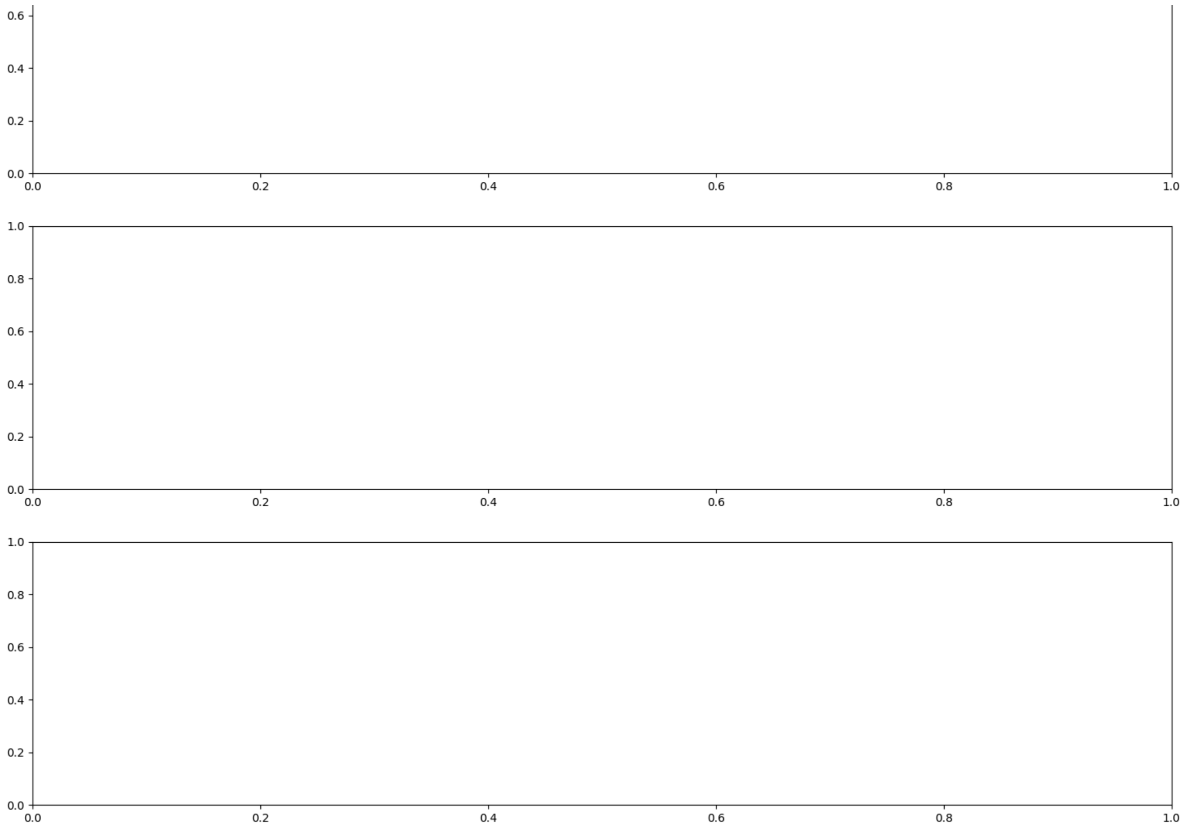
symmetry_worst vs Diagnosis



fractal_dimension_worst vs Diagnosis







In [209…

```
# Now let us to remove ouylier
for col in df.columns:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    l_bound = Q1 - 1.5 * IQR
    u_bound = Q3 + 1.5 * IQR
    df = df[(df[col] >= l_bound) & (df[col] <= u_bound)]
```

In [210…

```
df
```

Out[210…

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smooth
19	8510426	0	13.540	14.36	87.46	566.3	
20	8510653	0	13.080	15.71	85.63	520.0	
21	8510824	0	9.504	12.44	60.34	273.9	
37	854941	0	13.030	18.42	82.61	523.8	
40	855167	1	13.440	21.58	86.18	563.0	
...	...	...	...	...	...	...	
551	923780	0	11.130	22.44	71.49	378.4	
552	924084	0	12.770	29.43	81.35	507.9	
554	924632	0	12.880	28.92	82.50	514.3	
555	924934	0	10.290	27.61	65.67	321.4	
560	925292	0	14.050	27.15	91.38	600.4	

233 rows × 32 columns

After some preprocessing based on the KNN assumption , let us to start modeling

In [211…

```
x = df.drop(['id', 'diagnosis'], axis=1)
y = df['diagnosis']
```

In [212…

```
x_train, x_temp, y_train, y_temp = train_test_split(x, y, test_size=0.4, random_sta
x_val, x_test, y_val, y_test = train_test_split(x_temp, y_temp, test_size=0.5, rand
```

In [213…

```
# Scale the features
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_val_scaled = scaler.transform(x_val)
x_test_scaled = scaler.transform(x_test)
```

In [214…

```
cols=x.columns
```

In [215…

```
x_train = pd.DataFrame(x_train_scaled, columns=cols, index=x_train.index)
x_val = pd.DataFrame(x_val_scaled, columns=cols, index=x_val.index)
x_test = pd.DataFrame(x_test_scaled, columns=cols, index=x_test.index)
```

In [216…

```
# fitting
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(x_train, y_train)
```

Out[216…

```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```

In [217…

```
y_pred = knn.predict(x_val)
print(f"Initial validation accuracy: {accuracy_score(y_val, y_pred):.4f}")
```

```
Initial validation accuracy: 0.9362
```

In [218…

```
df
```

Out[218…

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smooth
19	8510426	0	13.540	14.36	87.46	566.3	
20	8510653	0	13.080	15.71	85.63	520.0	
21	8510824	0	9.504	12.44	60.34	273.9	
37	854941	0	13.030	18.42	82.61	523.8	
40	855167	1	13.440	21.58	86.18	563.0	
...	...	...	...	...	...	...	
551	923780	0	11.130	22.44	71.49	378.4	
552	924084	0	12.770	29.43	81.35	507.9	
554	924632	0	12.880	28.92	82.50	514.3	
555	924934	0	10.290	27.61	65.67	321.4	
560	925292	0	14.050	27.15	91.38	600.4	

233 rows × 32 columns

In [219…

```
# df_cross = df.drop('id', axis=1)
# df_cross['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})
```

In [220…

```
X_cross = df.drop(['id', 'diagnosis'], axis=1)
y_cross = df['diagnosis']
```

In [221…

```
X_scaled = scaler.fit_transform(X_cross)
```

In [222…

```
X_train, X_test, y_train_alt, y_test_alt = train_test_split(X_scaled, y_cross, test
```

In [223…

```
knn_alt = KNeighborsClassifier(n_neighbors=20)
cv_scores = cross_val_score(knn_alt, X_train, y_train_alt, cv=5)
```

the train here is almost = test so is no oferfiting

In [224…

```
knn_alt.fit(X_train, y_train_alt)
train_acc = accuracy_score(y_train_alt, knn_alt.predict(X_train))
test_acc = accuracy_score(y_test_alt, knn_alt.predict(X_test))
print(f"Train accuracy: {train_acc:.4f}")
print(f"Test accuracy: {test_acc:.4f}")
```

```
Train accuracy: 0.9462
Test accuracy: 0.9149
```

Testing different K values and selecting the optimal value

In [225…

```
k_values = range(1, 31)
train_accuracies = []
val_accuracies = []
test_accuracies = []
```

In [226…

```
for k in k_values:
    knn_model = KNeighborsClassifier(n_neighbors=k)

    knn_model.fit(x_train, y_train)

    train_accuracy = accuracy_score(y_train, knn_model.predict(x_train))
    train_accuracies.append(train_accuracy)

    val_accuracy = accuracy_score(y_val, knn_model.predict(x_val))
    val_accuracies.append(val_accuracy)

    test_accuracy = accuracy_score(y_test, knn_model.predict(x_test))
    test_accuracies.append(test_accuracy)
```

In [227…

```
print("Accuracy results for different K values:")
for i, k in enumerate(k_values):
    print(f"K = {k}: Train Accuracy = {train_accuracies[i]:.4f}, Validation Accurac
```

```
Accuracy results for different K values:
K = 1: Train Accuracy = 1.0000, Validation Accuracy = 0.9574, Test Accuracy = 0.9574
K = 2: Train Accuracy = 0.9784, Validation Accuracy = 0.9574, Test Accuracy = 0.8936
K = 3: Train Accuracy = 0.9784, Validation Accuracy = 0.9787, Test Accuracy = 0.9149
K = 4: Train Accuracy = 0.9712, Validation Accuracy = 0.9362, Test Accuracy = 0.8936
K = 5: Train Accuracy = 0.9712, Validation Accuracy = 0.9362, Test Accuracy = 0.8936
K = 6: Train Accuracy = 0.9712, Validation Accuracy = 0.9362, Test Accuracy = 0.8936
K = 7: Train Accuracy = 0.9712, Validation Accuracy = 0.9362, Test Accuracy = 0.8936
K = 8: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.8936
K = 9: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.8936
K = 10: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.893
6
K = 11: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.893
6
K = 12: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.893
6
K = 13: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.893
6
K = 14: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.893
6
K = 15: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.893
6
K = 16: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.893
6
K = 17: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.893
6
K = 18: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.893
6
K = 19: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.893
6
K = 20: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.893
6
K = 21: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.893
6
K = 22: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.893
6
K = 23: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.893
6
K = 24: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.893
6
K = 25: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.893
6
K = 26: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.893
6
K = 27: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.893
6
K = 28: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.893
6
K = 29: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.893
6
K = 30: Train Accuracy = 0.9568, Validation Accuracy = 0.9362, Test Accuracy = 0.893
6
```

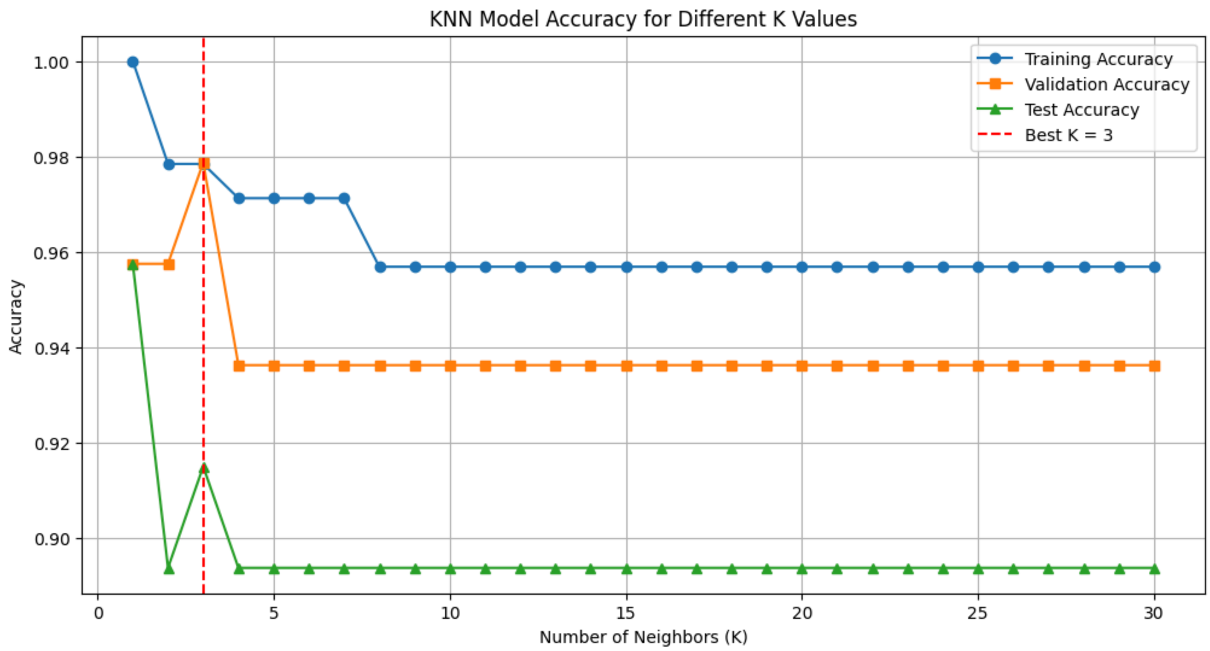
In [228…

```python
best_k_index = np.argmax(val_accuracies)
best_k = k_values[best_k_index]
print(f"\nBest K value is {best_k} with validation accuracy = {val_accuracies[best_
```

```
Best K value is 3 with validation accuracy = 0.9787
```

In [229...

```python
plt.figure(figsize=(12, 6))
plt.plot(k_values, train_accuracies, label='Training Accuracy', marker='o')
plt.plot(k_values, val_accuracies, label='Validation Accuracy', marker='s')
plt.plot(k_values, test_accuracies, label='Test Accuracy', marker='^')
plt.axvline(x=best_k, color='r', linestyle='--', label=f'Best K = {best_k}')
plt.xlabel('Number of Neighbors (K)')
plt.ylabel('Accuracy')
plt.title('KNN Model Accuracy for Different K Values')
plt.legend()
plt.grid(True)
plt.show()
```



Using the optimal K value and creating the final model

In [230...

```python
# Create model using the optimal K value
best_knn = KNeighborsClassifier(n_neighbors=best_k)
best_knn.fit(x_train, y_train)
```

Out[230...

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

Calculating and analyzing the confusion matrix

In [231...

```python
y_pred_test = best_knn.predict(x_test)
```

In [232...

```python
conf_matrix = confusion_matrix(y_test, y_pred_test)
```

In [233…
```
print("\nConfusion Matrix on Test Set:")
print(conf_matrix)
```

```
Confusion Matrix on Test Set:
[[42  0]
 [ 4  1]]
```

In [234…
```
tn, fp, fn, tp = conf_matrix.ravel()
print(f"\nTrue Positives (TP): {tp}")
print(f"True Negatives (TN): {tn}")
print(f"False Positives (FP): {fp}")
print(f"False Negatives (FN): {fn}")
```

```
True Positives (TP): 1
True Negatives (TN): 42
False Positives (FP): 0
False Negatives (FN): 4
```

Calculating performance metrics

In [235…
```
accuracy = accuracy_score(y_test, y_pred_test)
precision = precision_score(y_test, y_pred_test)
recall = recall_score(y_test, y_pred_test)
f1 = f1_score(y_test, y_pred_test)
```

In [236…
```
print("\nModel Performance Metrics on Test Set:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
```

```
Model Performance Metrics on Test Set:
Accuracy: 0.9149
Precision: 1.0000
Recall: 0.2000
F1-Score: 0.3333
```

In [237…
```
print("\nClassification Report on Test Set:")
print(classification_report(y_test, y_pred_test))
```

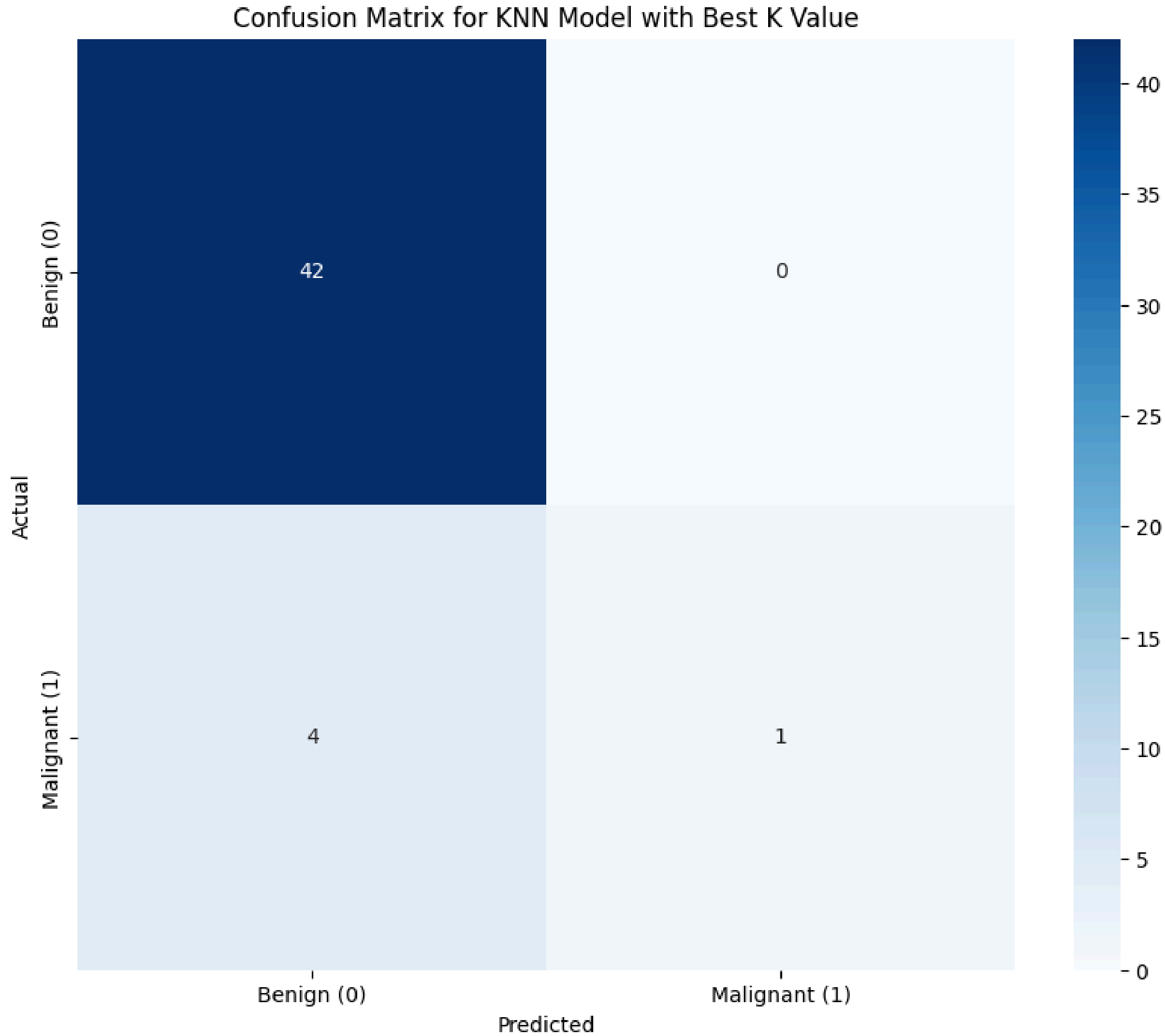
```
Classification Report on Test Set:
              precision    recall  f1-score   support

           0       0.91      1.00      0.95        42
           1       1.00      0.20      0.33         5

    accuracy                           0.91        47
   macro avg       0.96      0.60      0.64        47
weighted avg       0.92      0.91      0.89        47
```

In [238…
```
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Benign (0)', 'Malignant (1)'],
            yticklabels=['Benign (0)', 'Malignant (1)'])
```

```
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for KNN Model with Best K Value')
plt.show()
```



Applying Cross-Validation

In [239…

```
cv_scores = cross_val_score(best_knn, X_cross, y_cross, cv=5)
print("\nCross-Validation Scores (5-fold):")
print(f"Individual Scores: {cv_scores}")
print(f"Mean Accuracy: {cv_scores.mean():.4f}")
```

```
Cross-Validation Scores (5-fold):
Individual Scores: [0.95744681 0.95744681 0.95744681 1.         0.95652174]
Mean Accuracy: 0.9658
```

Comparing cross-validation performance with validation and test set performance

In [240…
```
print("\nPerformance Comparison:")
print(f"Cross-Validation Accuracy: {cv_scores.mean():.4f} ± {cv_scores.std():.4f}")
print(f"Validation Set Accuracy: {val_accuracies[best_k_index]:.4f}")
print(f"Test Set Accuracy: {accuracy:.4f}")
```

```
Performance Comparison:
Cross-Validation Accuracy: 0.9658 ± 0.0171
Validation Set Accuracy: 0.9787
Test Set Accuracy: 0.9149
```

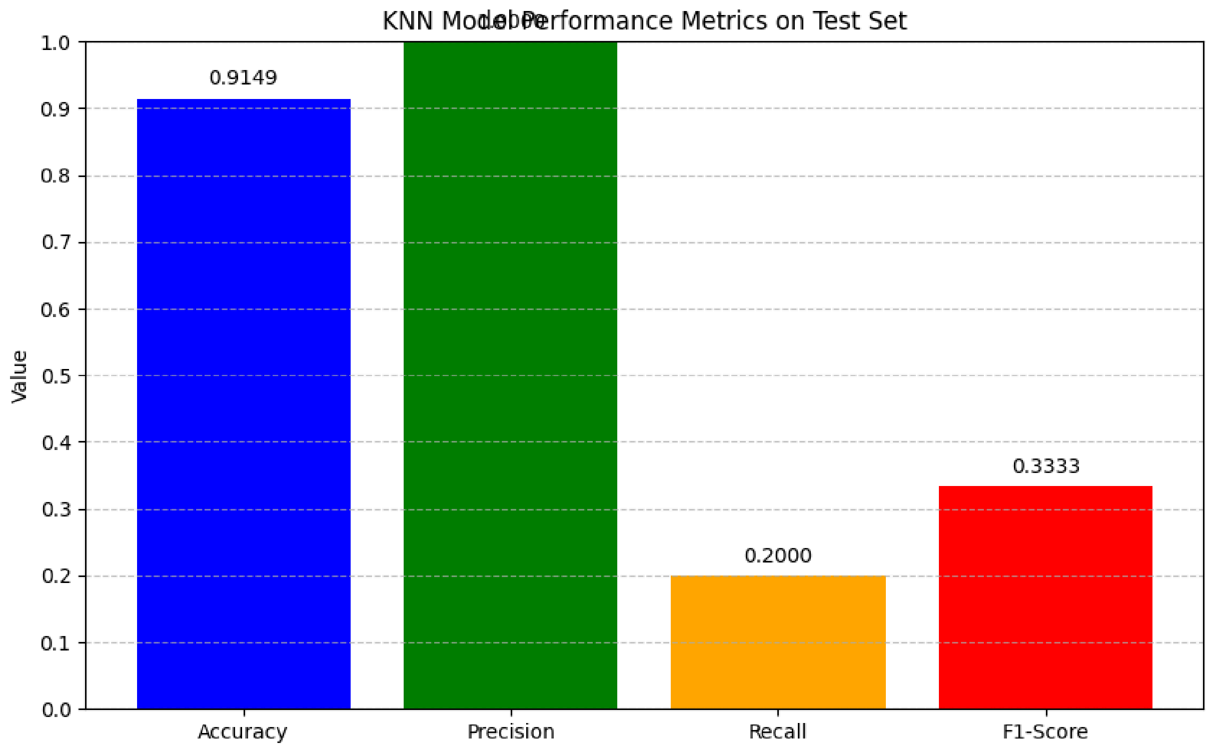
In [241…
```
if cv_scores.mean() > accuracy:
    print("\nCross-validation gives a higher performance estimate compared to the t
elif cv_scores.mean() < accuracy:
    print("\nCross-validation gives a lower performance estimate compared to the te
else:
    print("\nCross-validation and test set give similar performance estimates.")

print(f"The difference between cross-validation mean accuracy and test set accuracy
```

```
Cross-validation gives a higher performance estimate compared to the test set.
The difference between cross-validation mean accuracy and test set accuracy is 0.050
9
```

In [242…
```
metrics = ['Accuracy', 'Precision', 'Recall', 'F1-Score']
values = [accuracy, precision, recall, f1]
```

In [243…
```
plt.figure(figsize=(10, 6))
plt.bar(metrics, values, color=['blue', 'green', 'orange', 'red'])
plt.ylim(0, 1.0)
plt.yticks(np.arange(0, 1.1, 0.1))
plt.title('KNN Model Performance Metrics on Test Set')
plt.ylabel('Value')
for i, v in enumerate(values):
    plt.text(i, v + 0.02, f'{v:.4f}', ha='center')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



Discussion about Overfitting and Model Improvement

In [244…

```
print("Checking for Overfitting:")
print(f"Training Accuracy: {train_accuracies[best_k_index]:.4f}")
print(f"Validation Accuracy: {val_accuracies[best_k_index]:.4f}")
print(f"Test Accuracy: {test_accuracies[best_k_index]:.4f}")
```

```
Checking for Overfitting:
Training Accuracy: 0.9784
Validation Accuracy: 0.9787
Test Accuracy: 0.9149
```

In [245…

```
overfitting_threshold = 0.05
train_test_diff = train_accuracies[best_k_index] - test_accuracies[best_k_index]
```

In [246…

```
print(f"\nDifference between Training and Test Accuracy: {train_test_diff:.4f}")
```

```
Difference between Training and Test Accuracy: 0.0635
```

In [247…

```
if train_test_diff > overfitting_threshold:
    print("\nThe model suffers from overfitting - much better performance on traini
    print("\nTechniques to reduce overfitting:")
    print("1. Increase K value to reduce the impact of outliers")
    print("2. Select more important features and reduce dimensionality")
    print("3. Use regularization methods like dimensionality reduction")
    print("4. Use cross-validation for more accurate performance estimation")
else:
    print("\nThe model does not suffer from overfitting - the difference between tr
```

```
The model suffers from overfitting - much better performance on training set compare
d to test set

Techniques to reduce overfitting:
1. Increase K value to reduce the impact of outliers
2. Select more important features and reduce dimensionality
3. Use regularization methods like dimensionality reduction
4. Use cross-validation for more accurate performance estimation
```