Exploratory Data Analysis

In this section, we focus on understanding the structure of the dataset before proceeding with preprocessing or modeling steps. The primary goal is to identify the most significant features, given that the Wisconsin Breast Cancer Dataset is relatively high-dimensional (30 features, including 10 discriminative features each with 3 variant types).

Overview of EDA Process

Section I: Initial Data Audit and Info

In this section, we load the configuration files for this notebook and import the Wisconsin Breast Cancer dataset using pandas (CSV format). We then examine basic information about the dataset's structure and its features. Next, we check for the presence of any missing or null values. Finally, we analyze the distribution of the target labels to detect any class imbalance.

In [1]:

```
# Importing all of the necessary libraries for the EDA process
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import yaml

# Loading the configuration files for the notebook
with open("../notebook_config.yaml", 'r') as f:
    config = yaml.safe_load(f)
dataset_path = config['paths']['dataset_path']
```

In [2]:

```
# Loading the dataset using a Pandas Dataframe
df = pd.read_csv(dataset_path)
print(df.info())
print(df.describe())
```

```
<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
None
                 id  radius_mean  texture_mean  perimeter_mean    area_mean  \
count  5.690000e+02   569.000000    569.000000      569.000000   569.000000
mean   3.037183e+07    14.127292     19.289649       91.969033   654.889104
std    1.250206e+08     3.524049      4.301036       24.298981   351.914129
min    8.670000e+03     6.981000      9.710000       43.790000   143.500000
25%    8.692180e+05    11.700000     16.170000       75.170000   420.300000
50%    9.060240e+05    13.370000     18.840000       86.240000   551.100000
75%    8.813129e+06    15.780000     21.800000      104.100000   782.700000
max    9.113205e+08    28.110000     39.280000      188.500000  2501.000000

       smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
count       569.000000        569.000000      569.000000           569.000000
mean          0.096360          0.104341        0.088799             0.048919
std           0.014064          0.052813        0.079720             0.038803
min           0.052630          0.019380        0.000000             0.000000
25%           0.086370          0.064920        0.029560             0.020310
50%           0.095870          0.092630        0.061540             0.033500
75%           0.105300          0.130400        0.130700             0.074000
max           0.163400          0.345400        0.426800             0.201200
```

```
       symmetry_mean  ...  texture_worst  perimeter_worst   area_worst  \
count     569.000000  ...     569.000000       569.000000   569.000000   
mean        0.181162  ...      25.677223       107.261213   880.583128   
std         0.027414  ...       6.146258        33.602542   569.356993   
min         0.106000  ...      12.020000        50.410000   185.200000   
25%         0.161900  ...      21.080000        84.110000   515.300000   
50%         0.179200  ...      25.410000        97.660000   686.500000   
75%         0.195700  ...      29.720000       125.400000  1084.000000   
max         0.304000  ...      49.540000       251.200000  4254.000000   

       smoothness_worst  compactness_worst  concavity_worst  \
count        569.000000         569.000000       569.000000   
mean           0.132369           0.254265         0.272188   
std            0.022832           0.157336         0.208624   
min            0.071170           0.027290         0.000000   
25%            0.116600           0.147200         0.114500   
50%            0.131300           0.211900         0.226700   
75%            0.146000           0.339100         0.382900   
max            0.222600           1.058000         1.252000   

       concave points_worst  symmetry_worst  fractional_dimension_worst  \
count            569.000000      569.000000               569.000000   
mean               0.114606        0.290076                 0.083946   
std                0.065732        0.061867                 0.018061   
min                0.000000        0.156500                 0.055040   
25%                0.064930        0.250400                 0.071460   
50%                0.099930        0.282200                 0.080040   
75%                0.161400        0.317900                 0.092080   
max                0.291000        0.663800                 0.207500   

       Unnamed: 32  
count          0.0  
mean           NaN  
std            NaN  
min            NaN  
25%            NaN  
50%            NaN  
75%            NaN  
max            NaN  

[8 rows x 32 columns]
```

In [3]:
```python
# Checking for NULL values and NULL columns if any
print("Null Value / Missing Value Check", end=f"\n{'='*40}\n")
print(df.isnull().value_counts())

# Dropping the "Unnamed: 32", "id" column as they will not contribute to the EDA
df = df.drop(columns=['Unnamed: 32', 'id'], axis=1)
```

```
Null Value / Missing Value Check
========================================
id     diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  smoothnes
s_mean  compactness_mean  concavity_mean  concave points_mean  symmetry_mean  fra
ctal_dimension_mean  radius_se  texture_se  perimeter_se  area_se  smoothness_se  
compactness_se  concavity_se  concave points_se  symmetry_se  fractal_dimension_s
e  radius_worst  texture_worst  perimeter_worst  area_worst  smoothness_worst  co
mpactness_worst  concavity_worst  concave points_worst  symmetry_worst  fractal_d
imension_worst  Unnamed: 32
False  False      False        False         False           False      False    
False             False           False                False          False       
False       False        False        False     False          False          Fals
e         False              False        False              False          Fal
se          False            False        False              False          Fal
se            False              False          False                    True
569
Name: count, dtype: int64
```

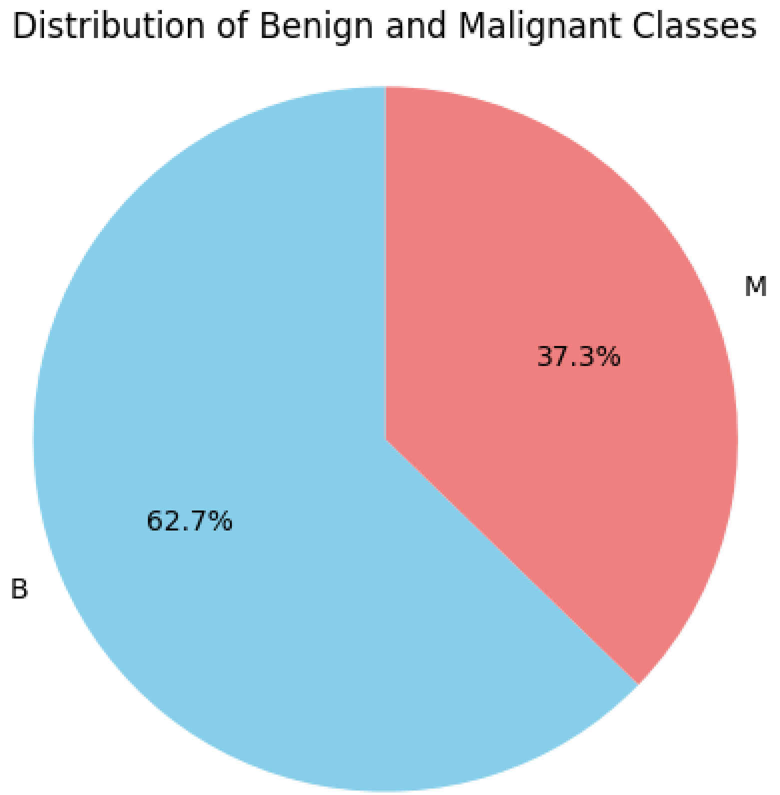
In [4]:

```python
# Visualizing the counts of "Benign" and "Malignant" classes in the dataset
class_counts = df['diagnosis'].value_counts()

print(class_counts)

# Plotting the Pie Chart
plt.figure(figsize=(5, 5))
plt.pie(class_counts, labels=class_counts.index, autopct='%1.1f%%', startangle=9
plt.title('Distribution of Benign and Malignant Classes')
plt.axis('equal')
plt.show()
```

```
diagnosis
B    357
M    212
Name: count, dtype: int64
```



Conclusion

- The Breast Cancer Wisconsin Dataset is a well-prepared dataset with a clearly defined structure and a large feature pool suitable for complex modeling techniques.
- All features, except the target label (diagnosis), are continuous variables, which simplifies the choice and design of modeling approaches. Additionally, the class distribution exhibits only a mild to moderate imbalance.
- Given the critical nature of predicting malignancy, careful model selection and feature selection will be important. This also motivates exploring appropriate sampling techniques in the preprocessing stage to address any class imbalance effectively.

Section II: Skewness and Kurtosis Analysis

In this section, we analyze the skewness and kurtosis of each feature to better understand their distributions.

In [5]:
```python
# Creating a copy of the dataset and splitting as features and target label
bcd = df.copy()
X = bcd.drop(columns=['diagnosis'])
y = bcd['diagnosis']

# Calculating the Skewness and Kurtosis of the features involved
skewness = X.skew().sort_values(ascending=False)
kurtosis = X.kurtosis().sort_values(ascending=False)

# Skewness
print("Skewness in the data")
print(skewness)
print("="*20)

# Kurtosis
print("Kurtosis in the data")
print(kurtosis)
print("*"*20)
```

```
Skewness in the data
area_se                    5.447186
concavity_se               5.110463
fractal_dimension_se       3.923969
perimeter_se               3.443615
radius_se                  3.088612
smoothness_se              2.314450
symmetry_se                2.195133
compactness_se             1.902221
area_worst                 1.859373
fractal_dimension_worst    1.662579
texture_se                 1.646444
area_mean                  1.645732
compactness_worst          1.473555
concave points_se          1.444678
symmetry_worst             1.433928
concavity_mean             1.401180
fractal_dimension_mean     1.304489
compactness_mean           1.190123
concave points_mean        1.171180
concavity_worst            1.150237
perimeter_worst            1.128164
radius_worst               1.103115
perimeter_mean             0.990650
radius_mean                0.942380
symmetry_mean              0.725609
texture_mean               0.650450
texture_worst              0.498321
concave points_worst       0.492616
smoothness_mean            0.456324
smoothness_worst           0.415426
dtype: float64
====================
Kurtosis in the data
area_se                    49.209077
concavity_se               48.861395
fractal_dimension_se       26.280847
perimeter_se               21.401905
radius_se                  17.686726
smoothness_se              10.469840
symmetry_se                 7.896130
texture_se                  5.349169
fractal_dimension_worst     5.244611
concave points_se           5.126302
compactness_se              5.106252
symmetry_worst              4.444560
area_worst                  4.396395
area_mean                   3.652303
compactness_worst           3.039288
fractal_dimension_mean      3.005892
concavity_mean              1.998638
compactness_mean            1.650130
concavity_worst             1.615253
symmetry_mean               1.287933
perimeter_worst             1.070150
concave points_mean         1.066556
perimeter_mean              0.972214
radius_worst                0.944090
smoothness_mean             0.855975
radius_mean                 0.845522
```

```
texture_mean               0.758319
smoothness_worst           0.517825
texture_worst              0.224302
concave points_worst      -0.535535
dtype: float64
********************
```

Conclusion

- Skewness measures the asymmetry of a data distribution, indicating whether values tend to lean left or right of the mean. Kurtosis measures the "tailedness" or peakedness, revealing the presence of outliers or extreme values.
- Our data exhibits significant skewness and kurtosis, reflecting meaningful outliers that are typical in medical datasets such as cancer data. Although these metrics suggest deviation from normality, it is important not to remove outliers blindly, as they carry clinical significance.
- We will further visualize and analyze these patterns to devise an effective modeling strategy that respects the underlying data distribution and its clinical implications.

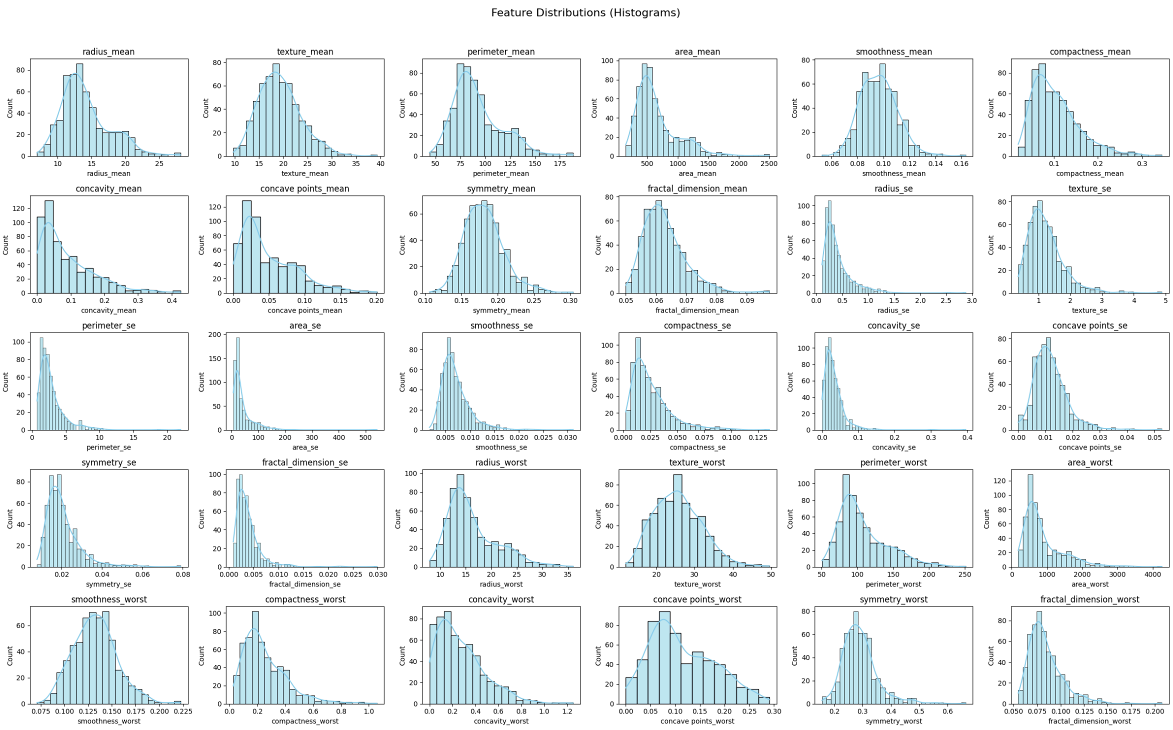
Section III: Univariate Analysis

In this section, we analyze the feature distributions using visualizations to identify useful patterns and reinforce the findings from the previous section. We start by examining histograms to better understand the distribution of each feature. Finally, we conclude this section by detecting outliers through box plots.

In [6]:

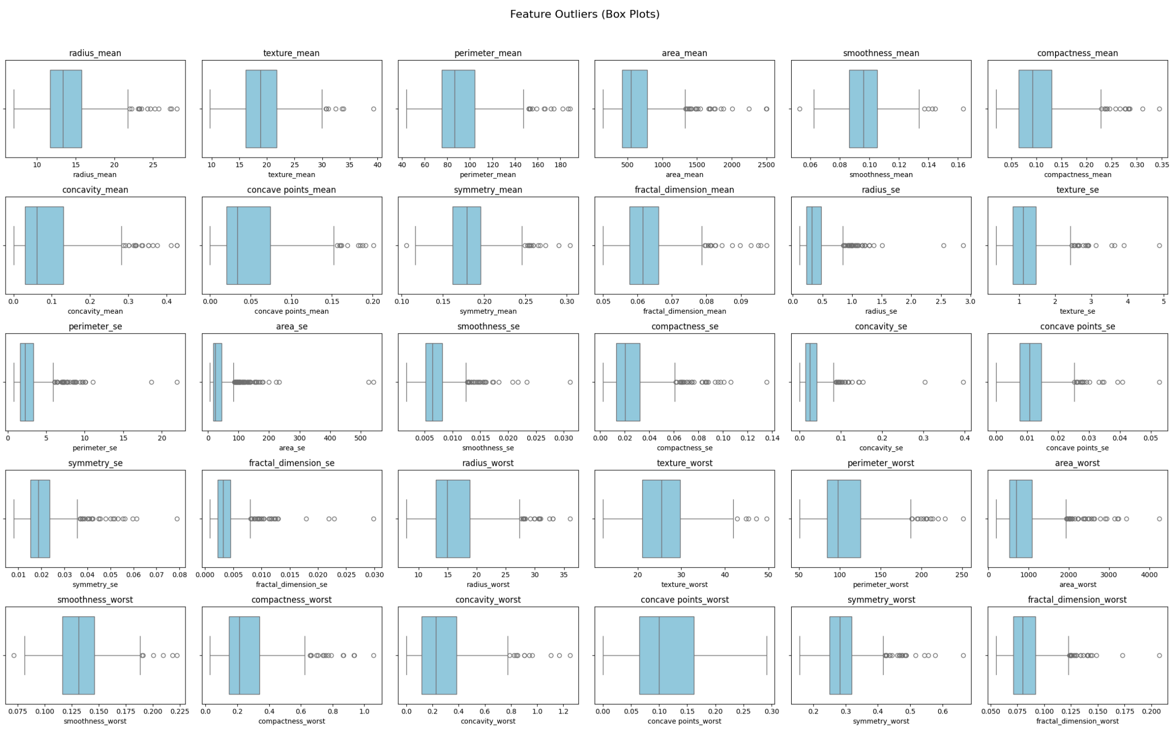
```
# Histograms for Feature distribution
import math
# Number of features
num_features = X.shape[1]
# Dynamically calculate rows and columns for subplots
cols = 6
rows = math.ceil(num_features / cols)
# Histograms grid
fig, axes = plt.subplots(rows, cols, figsize=(cols*4, rows*3))
axes = axes.flatten()

for i, col in enumerate(X.columns):
    sns.histplot(X[col], kde=True, ax=axes[i], color='skyblue')
    axes[i].set_title(col)
# Hide empty subplots if any
for j in range(i+1, len(axes)):
    axes[j].axis('off')
plt.suptitle('Feature Distributions (Histograms)', fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```



In [7]:

```
# Box plots for outlier detection
# Box plots grid
fig, axes = plt.subplots(rows, cols, figsize=(cols*4, rows*3))
axes = axes.flatten()
for i, col in enumerate(X.columns):
    sns.boxplot(x=X[col], ax=axes[i], color='skyblue')
    axes[i].set_title(col)
# Hide empty subplots if any
for j in range(i+1, len(axes)):
    axes[j].axis('off')
plt.suptitle('Feature Outliers (Box Plots)', fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```



Conclusion

- The analysis clearly indicates that the dataset contains a significant number of outliers, which may be meaningful given the nature of tumor malignancy data. The presence of heavy tails suggests that careful handling will be required during preprocessing and modeling.
- Fortunately, most features exhibit a Gaussian-like distribution with only low to moderate skewness.

Section IV: Bivariate Analysis

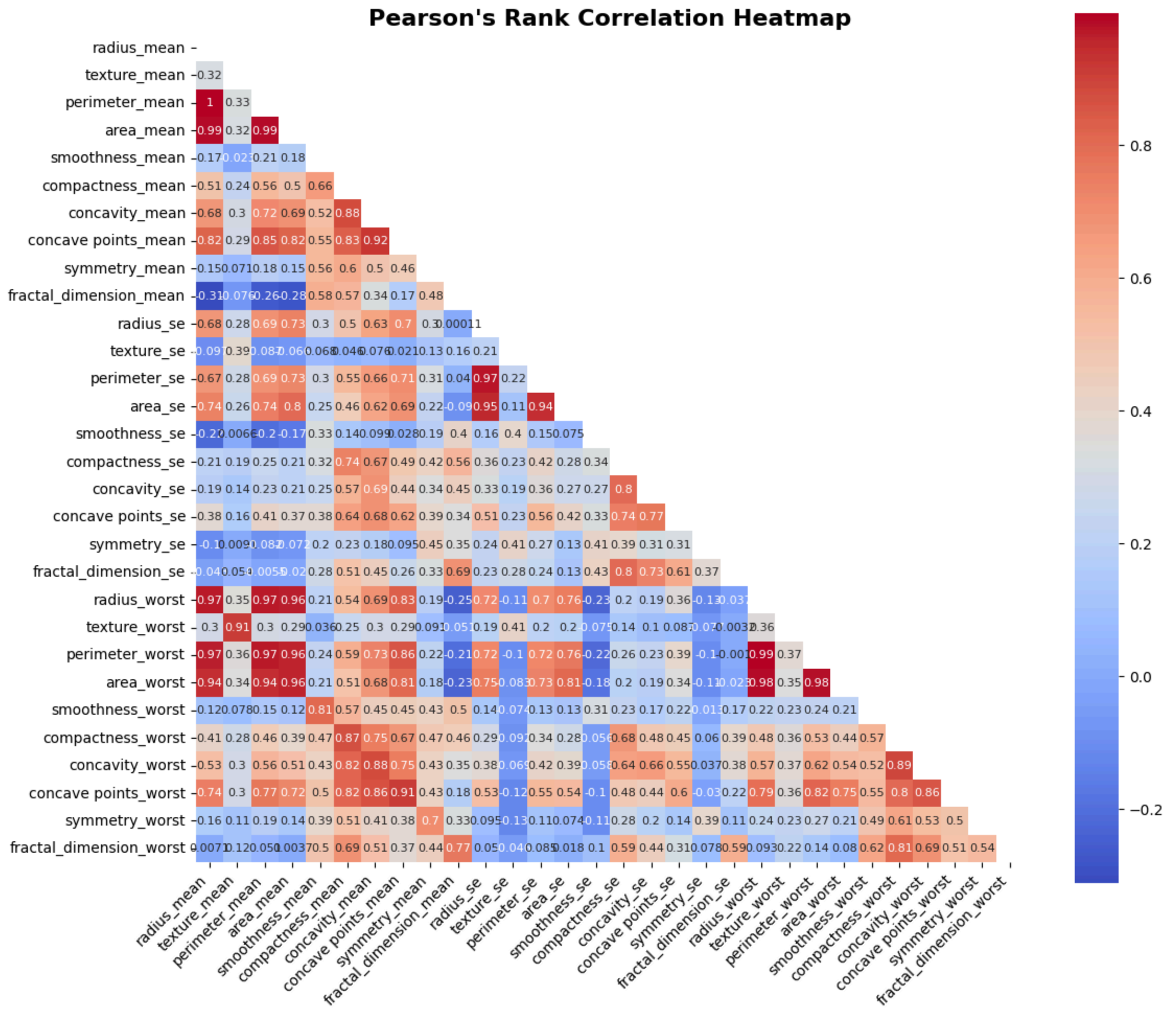
In this section, we focus on understanding the relationships between variables and between features and the target label to gain insights for feature engineering and selection.

First, we apply Pearson's correlation to detect multicollinearity and redundancy among features. Then, we use point biserial correlation to identify which features contribute most significantly to the target label.

In [8]:
```
# Pearson's Rank correlation
p_corr = X.corr(method="pearson")

mask = np.triu(np.ones_like(p_corr, dtype=bool))

plt.figure(figsize=(12, 10))
sns.heatmap(p_corr, cmap='coolwarm', annot=True, annot_kws={"size": 8}, square=T
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.yticks(rotation=0, fontsize=10)
plt.title("Pearson's Rank Correlation Heatmap", fontsize=16, weight='bold')
plt.tight_layout()
plt.show()
```



In [9]:
```python
# Point biserial correlation
from scipy.stats import pointbiserialr
y_binary = y.map({'B': 0, 'M': 1})

correlations = {}
p_values = {}

for col in X.columns:
    corr, p_val = pointbiserialr(X[col], y_binary)
    correlations[col] = corr
    p_values[col] = p_val

corr_df = pd.DataFrame({'Correlation': correlations, 'P-Value': p_values})
print(corr_df.sort_values(by='Correlation', ascending=False))
```

```
                          Correlation        P-Value
concave points_worst         0.793566  1.969100e-124
perimeter_worst              0.782914  5.771397e-119
concave points_mean          0.776614  7.101150e-116
radius_worst                 0.776454  8.482292e-116
perimeter_mean               0.742636  8.436251e-101
area_worst                   0.733825   2.828848e-97
radius_mean                  0.730029   8.465941e-96
area_mean                    0.708984   4.734564e-88
concavity_mean               0.696360   9.966556e-84
concavity_worst              0.659610   2.464664e-72
compactness_mean             0.596534   3.938263e-56
compactness_worst            0.590998   7.069816e-55
radius_se                    0.567134   9.738949e-50
perimeter_se                 0.556141   1.651905e-47
area_se                      0.548236   5.895521e-46
texture_worst                0.456903   1.078057e-30
smoothness_worst             0.421465   6.575144e-26
symmetry_worst               0.416294   2.951121e-25
texture_mean                 0.415185   4.058636e-25
concave points_se            0.408042   3.072309e-24
smoothness_mean              0.358560   1.051850e-18
symmetry_mean                0.330499   5.733384e-16
fractal_dimension_worst      0.323872   2.316432e-15
compactness_se               0.292999   9.975995e-13
concavity_se                 0.253730   8.260176e-10
fractal_dimension_se         0.077972   6.307355e-02
symmetry_se                 -0.006522   8.766418e-01
texture_se                  -0.008303   8.433320e-01
fractal_dimension_mean      -0.012838   7.599368e-01
smoothness_se               -0.067016   1.102966e-01
```

Conclusion

The results clearly indicate significant correlations within two main groups of features:

- Group 1: Area, Perimeter, and Radius
- Group 2: Concave Points, Compactness, and Concavity

This collinearity suggests challenges for linear models in classification, emphasizing the importance of careful feature selection. Some features may need to be dropped or combined to create new, more informative features.

Additionally, point biserial correlation reveals that the 'worst' variant features are the most significant, which aligns with the biological nature of tumor malignancy.

Section V: Dimensionality Reduction and Class Boundary Visualization

In this section, we apply t-SNE for nonlinear dimensionality reduction. This technique preserves local neighborhood structures while condensing the features into a few components, making patterns easier to visualize.

Using t-SNE, we examine how samples cluster, observing whether different classes form distinct groups or overlap. This also helps us assess whether the current features carry sufficient signal to separate the classes effectively.

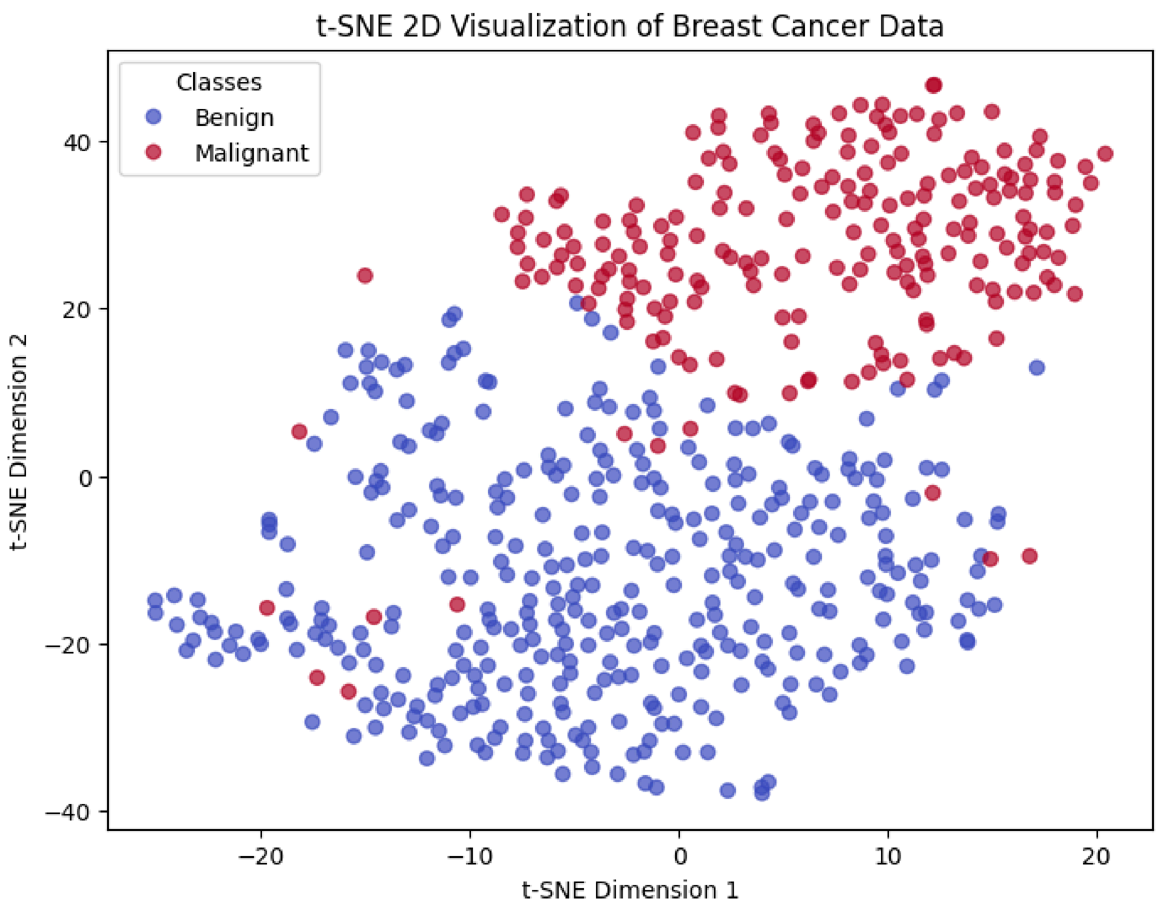
In [10]:
```
# TSNE Calculation
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

tsne_2d = TSNE(n_components=2, random_state=42, perplexity=30, max_iter=1200, ea
X_tsne_2d = tsne_2d.fit_transform(X_scaled)
```

In [11]:
```
# TSNE Visualization
target = y.map({'B':0, 'M':1}).values  # convert to numeric if needed

plt.figure(figsize=(8,6))
scatter = plt.scatter(X_tsne_2d[:,0], X_tsne_2d[:,1], c=target, cmap='coolwarm',
handles, _ = scatter.legend_elements()
plt.legend(handles, ['Benign', 'Malignant'], title="Classes")
plt.title("t-SNE 2D Visualization of Breast Cancer Data")
plt.xlabel("t-SNE Dimension 1")
plt.ylabel("t-SNE Dimension 2")
plt.show()
```



Conclusion

The t-SNE visualization reveals clearly separated clusters corresponding to benign and malignant classes, indicating that the features effectively capture class distinctions. A few malignant samples overlap with benign ones, suggesting borderline cases or subtle feature similarities.

Despite box plots indicating outliers, their absence as extreme points in the t-SNE plot implies these outliers represent genuine variations and meaningful rare cases rather than noise. This supports preserving them during modeling to retain critical clinical information.

Section VI: Feature Engineering Ideas

While the current feature set effectively distinguishes between classes, our models may benefit from additional domain-informed engineered features.

We will augment the existing feature pool with these new features and evaluate their contribution using point biserial correlation. To further validate their impact, we will perform t-SNE on the filtered features that exceed a specified threshold (e.g., 0.5) based on point biserial correlation rankings.

In [12]:

```
# Some Feature engineering before further analysis
X_c = X.copy()
y_c = y.copy()

X_c['con_worst_per_area'] = X_c['concavity_worst'] / X_c['area_worst']
X_c['con_mean_per_area'] = X_c['concavity_mean'] / X_c['area_mean']

X_c['fd_worst_perimeter'] = X_c['fractal_dimension_worst']*X_c['perimeter_worst'
X_c['fd_mean_perimeter'] = X_c['fractal_dimension_mean']*X_c['perimeter_mean']

X_c['smooth_worst_radius'] = X_c['smoothness_worst']*X_c['radius_worst']
X_c['smooth_worst_radius'] = X_c['smoothness_mean']*X_c['radius_mean']
```

In [13]:

```
# Point Biserial Correlation Ranks
y_binary_eng = y_c.map({'B': 0, 'M': 1})
correlations = {}
p_values = {}
for col in X_c.columns:
    corr, p_val = pointbiserialr(X_c[col], y_binary_eng)
    correlations[col] = corr
    p_values[col] = p_val
eng_corr_df = pd.DataFrame({'Correlation': correlations, 'P-Value': p_values})
print(eng_corr_df.sort_values(by='Correlation', ascending=False))
```

```
                          Correlation        P-Value
concave points_worst         0.793566  1.969100e-124
perimeter_worst              0.782914  5.771397e-119
concave points_mean          0.776614  7.101150e-116
radius_worst                 0.776454  8.482292e-116
fd_worst_perimeter           0.768996  2.853425e-112
smooth_worst_radius          0.749640  1.043846e-103
fd_mean_perimeter            0.745731  4.496630e-102
perimeter_mean               0.742636  8.436251e-101
area_worst                   0.733825   2.828848e-97
radius_mean                  0.730029   8.465941e-96
area_mean                    0.708984   4.734564e-88
concavity_mean               0.696360   9.966556e-84
concavity_worst              0.659610   2.464664e-72
compactness_mean             0.596534   3.938263e-56
compactness_worst            0.590998   7.069816e-55
radius_se                    0.567134   9.738949e-50
perimeter_se                 0.556141   1.651905e-47
area_se                      0.548236   5.895521e-46
texture_worst                0.456903   1.078057e-30
smoothness_worst             0.421465   6.575144e-26
symmetry_worst               0.416294   2.951121e-25
texture_mean                 0.415185   4.058636e-25
concave points_se            0.408042   3.072309e-24
smoothness_mean              0.358560   1.051850e-18
symmetry_mean                0.330499   5.733384e-16
fractal_dimension_worst      0.323872   2.316432e-15
compactness_se               0.292999   9.975995e-13
con_mean_per_area            0.269681   6.139523e-11
concavity_se                 0.253730   8.260176e-10
con_worst_per_area           0.083399   4.676053e-02
fractal_dimension_se         0.077972   6.307355e-02
symmetry_se                 -0.006522   8.766418e-01
texture_se                  -0.008303   8.433320e-01
fractal_dimension_mean      -0.012838   7.599368e-01
smoothness_se               -0.067016   1.102966e-01
```

In [14]:

```
# Separating point biserial correlation ranks for Raw vs Raw + Engineered Featur

threshold = 0.25
# Now we have the initial pool of features one where engineered features are pre
filtered_eng_corr_df = eng_corr_df[eng_corr_df['Correlation'].abs() >= threshold
filtered_corr_df = corr_df[corr_df['Correlation'].abs() >= threshold]

# Save as column names for ease of use
init_eng_feature_pool = filtered_eng_corr_df.index.to_list()
init_feature_pool = filtered_corr_df.index.to_list()
```

Conclusion

The point biserial correlation indicates that the newly engineered features are significant. However, we can further strengthen our analysis through downstream processing to select features more efficiently, reducing dimensionality while maintaining or even improving class representation.

Section VII: Feature Selection & Ranking

In this section, we apply a two-tier technique to efficiently select the most important and highest-ranking features from the feature pools using mRMR followed by a Random Forest approach.

We will then use t-SNE on these selected feature sets to evaluate whether the feature representation and class separability are maintained in both cases.

In [15]:
```
# mRMR test
import pymrmr

y_numeric = y.map({'B': 0, 'M': 1})

# For engineered pool
X_eng = X_c[init_eng_feature_pool]
df_eng = pd.concat([y_numeric, X_eng], axis=1)
df_eng.columns = ['target'] + list(X_eng.columns)

# For raw pool
X_raw = X[init_feature_pool]
df_raw = pd.concat([y_numeric, X_raw], axis=1)
df_raw.columns = ['target'] + list(X_raw.columns)

top_k = 25
# For engineered pool
mrmr_eng_feats = pymrmr.mRMR(df_eng, 'MIQ', top_k)
# For raw pool
mrmr_raw_feats = pymrmr.mRMR(df_raw, 'MIQ', top_k)
```

In [16]:
```
# Running through a Random Forest Classifier for final feature rank
from sklearn.ensemble import RandomForestClassifier

def run_rf_importance(X, y, n_select=15):
    clf = RandomForestClassifier(n_estimators=100, random_state=0)
    clf.fit(X, y)
    importances = pd.Series(clf.feature_importances_, index=X.columns)
    top_feats = importances.sort_values(ascending=False).head(n_select).index.tc
    return top_feats

X_eng_mrmr = X_c[mrmr_eng_feats]
X_raw_mrmr = X[mrmr_raw_feats]

top20_eng_feats_rf = run_rf_importance(X_eng_mrmr, y_numeric, n_select=20)
top20_raw_feats_rf = run_rf_importance(X_raw_mrmr, y_numeric, n_select=20)

print("Top 20 Engineered Pool Features (Random Forest):", top20_eng_feats_rf)
print("Top 20 Raw Pool Features (Random Forest):", top20_raw_feats_rf)

print(len(top20_eng_feats_rf))
print(len(top20_raw_feats_rf))
```

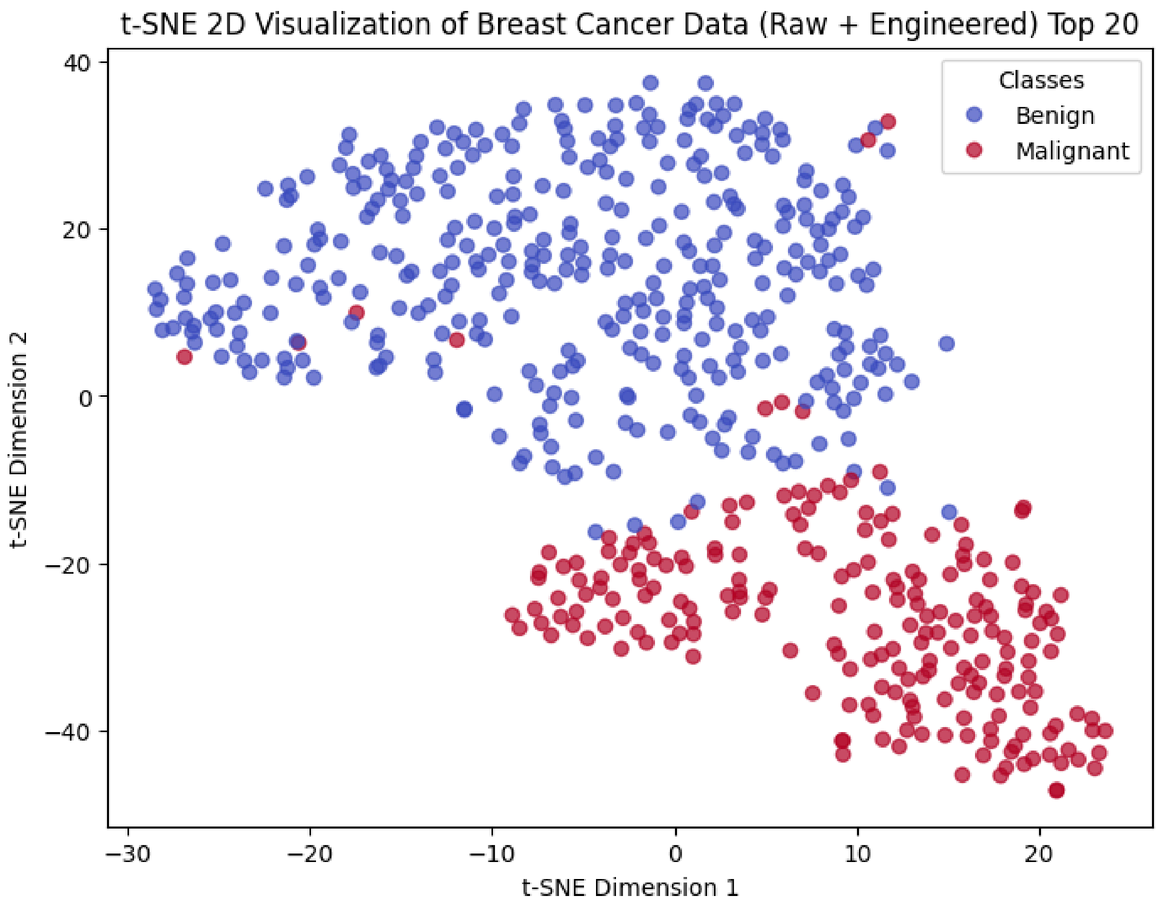
```
Top 20 Engineered Pool Features (Random Forest): ['fd_worst_perimeter', 'smooth_w
orst_radius', 'perimeter_worst', 'concave points_worst', 'radius_worst', 'area_wo
rst', 'fd_mean_perimeter', 'radius_mean', 'concavity_mean', 'perimeter_mean', 'ar
ea_se', 'area_mean', 'texture_worst', 'texture_mean', 'concavity_worst', 'smoothn
ess_worst', 'compactness_worst', 'symmetry_worst', 'fractal_dimension_worst', 'ra
dius_se']
Top 20 Raw Pool Features (Random Forest): ['perimeter_worst', 'concave points_wor
st', 'radius_worst', 'concave points_mean', 'area_worst', 'concavity_mean', 'peri
meter_mean', 'area_se', 'area_mean', 'radius_se', 'concavity_worst', 'radius_mea
n', 'texture_worst', 'compactness_worst', 'perimeter_se', 'smoothness_worst', 'te
xture_mean', 'symmetry_worst', 'smoothness_mean', 'concavity_se']
20
20
```

In [17]:
```python
# Visualizing t-SNE for Raw + Engineered Feature Set
X_eng_filtered = X_c[top20_eng_feats_rf]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_eng_filtered)

tsne_2d = TSNE(n_components=2, random_state=42, perplexity=30, max_iter=1200, ea
X_tsne_2d = tsne_2d.fit_transform(X_scaled)

# TSNE Visualization
target = y_c.map({'B':0, 'M':1}).values  # convert to numeric if needed

plt.figure(figsize=(8,6))
scatter = plt.scatter(X_tsne_2d[:,0], X_tsne_2d[:,1], c=target, cmap='coolwarm',
handles, _ = scatter.legend_elements()
plt.legend(handles, ['Benign', 'Malignant'], title="Classes")
plt.title(f"t-SNE 2D Visualization of Breast Cancer Data (Raw + Engineered) Top
plt.xlabel("t-SNE Dimension 1")
plt.ylabel("t-SNE Dimension 2")
plt.show()
```



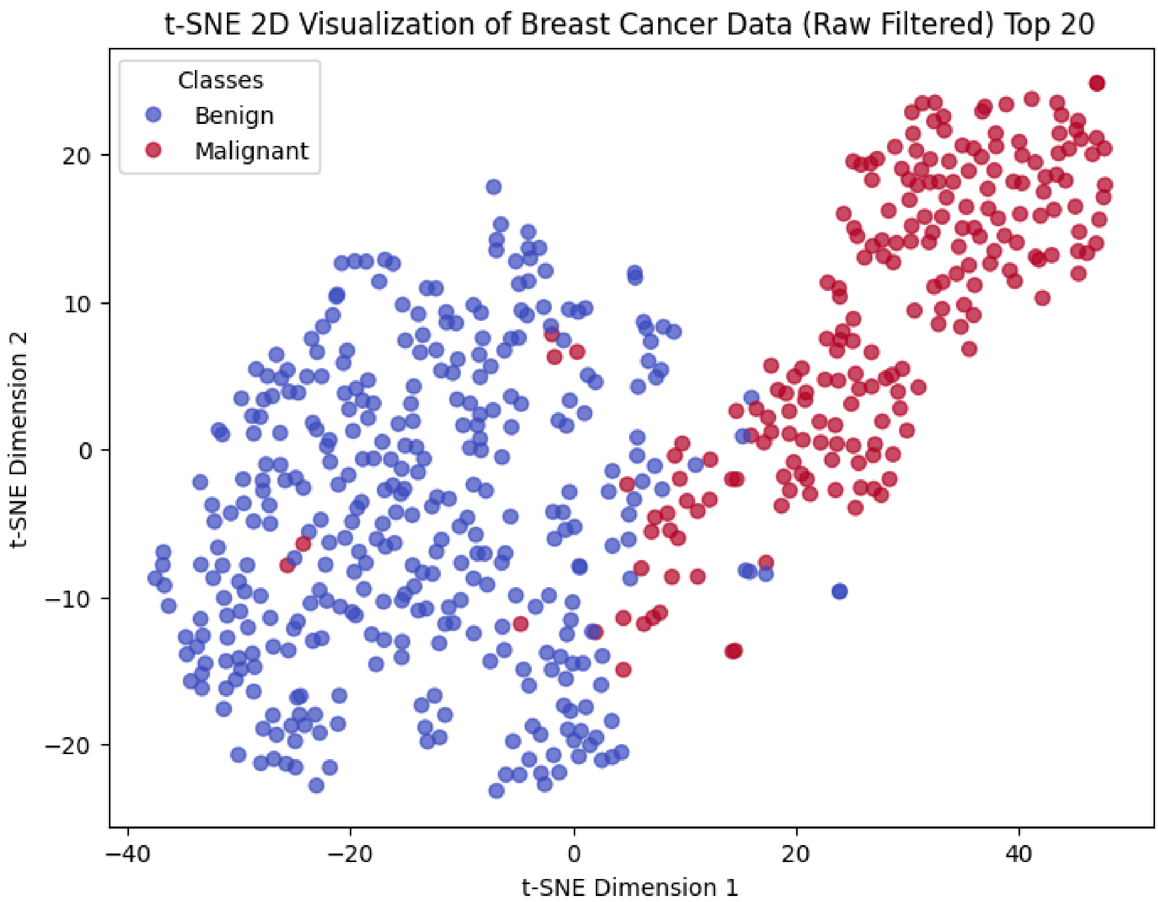
In [18]:

```
# Visualizing t-SNE for Raw Feature Set (Filtered by PBC ranks)
X_filtered = X[top20_raw_feats_rf]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_filtered)

tsne_2d = TSNE(n_components=2, random_state=42, perplexity=30, max_iter=1200, ea
X_tsne_2d = tsne_2d.fit_transform(X_scaled)

# TSNE Visualization
target = y.map({'B':0, 'M':1}).values  # convert to numeric if needed

plt.figure(figsize=(8,6))
scatter = plt.scatter(X_tsne_2d[:,0], X_tsne_2d[:,1], c=target, cmap='coolwarm',
handles, _ = scatter.legend_elements()
plt.legend(handles, ['Benign', 'Malignant'], title="Classes")
plt.title(f"t-SNE 2D Visualization of Breast Cancer Data (Raw Filtered) Top {len
plt.xlabel("t-SNE Dimension 1")
plt.ylabel("t-SNE Dimension 2")
plt.show()
```



Conclusion

Having successfully demonstrated that both lower-dimensional feature sets maintain comparable class separation, we can now proceed to the preprocessing step to efficiently utilize these feature sets in a modeling pipeline.

Section VIII: Finalizing Feature Pools and Findings

We have successfully completed the EDA process and uncovered several important insights:

- The Breast Cancer dataset exhibits low to moderate class imbalance, which is not expected to significantly impact modeling.
- Significant skewness and outliers are present in feature distributions; however, these outliers represent genuine and important information. Careful preprocessing is required to preserve their value.
- Many features are highly correlated, indicating that linear modeling may be challenging. Our two-tier feature selection strategy helps mitigate this by reducing redundancy.
- The t-SNE analysis demonstrated clear class separation when comparing all 30 features to the filtered features obtained from our two-tier selection.
- We effectively reduced dimensionality by nearly half while maintaining distinct class separation, enabling us to apply varied modeling techniques, with SVM being a natural candidate to explore.

Moving forward, we will save these selected feature sets and focus on preprocessing strategies tailored to each set. This will include building pipelines specific to the feature pools and exploring suitable modeling approaches and requirements for both cases.

In [19]:
```
import pickle

with open('../features/top20_eng_feats_rf.pkl', 'wb') as f:
    pickle.dump(top20_eng_feats_rf, f)

# Save raw features
with open('../features/top20_raw_feats_rf.pkl', 'wb') as f:
    pickle.dump(top20_raw_feats_rf, f)
```