In [76]:

```
from posixpath import split
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.neural_network import MLPRegressor
from sklearn.impute import SimpleImputer

import xgboost as xgb
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge, Lasso, BayesianRidge
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline

from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException

import time
import shutil
import os
```

In [77]:
```python
def calculate_k_bb(df):
    k_cols = []
    bb_cols = []
    for col in df.columns:
        print(col)
        if ("k%_" in col and "bb%_" not in col):
            k_cols.append(col)
        elif ("bb%_" in col and "k%_" not in col):
            bb_cols.append(col)
        else:
            print("NOT A MODEL OUTPUT COLUMN")
    df["k% mods"] = df[k_cols].mean(axis=1)
    df["bb% mods"] = df[bb_cols].mean(axis=1)
    df["k-bb act"] = df["k%"] - df["bb%"]
    df["k-bb mods"] = df["k% mods"] - df["bb% mods"]
    df["k-bb diff"] = df["k-bb mods"] - df["k-bb act"]
    df["w_k_bb"] = df["k% mods"]-0.3*df["bb% mods"]
    df["k/bb act"] = df["k%"] / df["bb%"]
    df["k/bb mods"] = df["k% mods"] / df["bb% mods"]
    df["k/bb diff"] = df["k/bb mods"] - df["k/bb act"]
    return df

def evaluate_model(model, X_test, y_test, target_variable):
    if 'xgb' in target_variable.lower():
        print("XGB MODEL")
    elif 'poly' in target_variable.lower():
        poly = model.named_steps['poly']
        model = model.named_steps['regressor']
        X_test = poly.transform(x_test)
    else:
        print("Model type not recognized try again")

    # Make predictions
    y_preds = model.predict(X_test)

    # Calculate evaluation metrics
    mse = mean_squared_error(y_test, y_preds)
    mae = mean_absolute_error(y_test, y_preds)
    r2 = r2_score(y_test, y_preds)

    # Print evaluation metrics
    print(f"Evaluation metrics for {target_variable}:")
    print(f"Model Type {model}:")
    print("Mean Squared Error (MSE):", mse)
    print("Mean Absolute Error (MAE):", mae)
    print("R-squared (R2) Score:", r2)
    print()

    return y_preds


def polynomial_learning(X, y, target_variable):
    # Pipeline definition
    pipeline = Pipeline([
        ('poly', PolynomialFeatures(include_bias=True)),
        ('regressor', Lasso())  # Initial regressor, will be replaced later
```

```
    ])

    # Parameter grid
    param_grid = {
        'regressor': [Lasso(), Ridge()],  # Regressors to try
        'poly__degree': [1, 2, 3, 4],  # Degrees to try
        'poly__interaction_only': [False],
        'regressor__alpha': [0.001, 0.01, 0.1, 0.25, 1.0, 2.5, 5, 10.0, 20]  #
Alphas to try
    }

    # GridSearchCV object
    model = GridSearchCV(pipeline, param_grid, scoring='neg_mean_squared_erro
r', n_jobs=-1, cv=4)

    # Fit the model
    model.fit(X, y)

    # Extract best estimator and regressor
    best_estimator = model.best_estimator_
    best_regressor = best_estimator.named_steps['regressor']

    # Determine the best regressor type
    if isinstance(best_regressor, Lasso):
        print(f"Best regressor for {target_variable}: Lasso")
    elif isinstance(best_regressor, Ridge):
        print(f"Best regressor for {target_variable}: Ridge")
    else:
        print(f"Unknown best regressor for {target_variable}")

    # Extract polynomial features and regressor
    poly = best_estimator.named_steps['poly']
    regress = best_estimator.named_steps['regressor']

    # Print best score
    print(f"BEST SCORE for {target_variable}: {-model.best_score_}")

    # Transform features using the best estimator
    X_poly = poly.transform(X)

    # Fit the model on polynomial features
    regress.fit(X_poly, y)

    # Print coefficients and best parameters
    print("Coefficients:", regress.coef_)
    print(f"Best training parameters for {target_variable}: ", regress.get_par
ams())

    #feature_names = poly.get_feature_names_out(input_features=X.columns)

    # Create a DataFrame to hold the feature names and their corresponding coe
fficients
    #coefficients_df = pd.DataFrame({
    #    'Feature': feature_names,
    #    'Coefficient': regress.coef_
    #})
```

```python
    #print(coefficients_df)

    return best_estimator

def split_data(df):
    df.columns = df.columns.str.lower()
    df.set_index(['season','name'],inplace=True)
    feat_cols = ['swstr%'] + list(df.loc[:,'stuff+':'cstr%'].columns)

    df[feat_cols] = df[feat_cols].replace('', np.nan)
    df.dropna(subset=feat_cols,inplace=True)
    print(df[df[feat_cols].isna().any(axis=1)])

    return df
```

In [78]:

```
'''from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time
import shutil
import os

# URL of the website
url = "https://www.fangraphs.com/leaders/major-league?pos=all&lg=all&type=c%2C
13%2C6%2C38%2C41%2C42%2C43%2C44%2C45%2C47%2C48%2C49%2C50%2C51%2C62%2C113%2C12
0%2C121%2C217%2C122%2C124%2C165%2C325%2C328%2C332%2C368%2C386%2C387%2C388%2C10
5%2C106%2C107%2C108%2C109%2C110%2C111%2C330%2C331&month=0&ind=1&rost=0&age=0&f
ilter=&players=0&startdate=&enddate=&season1=2024&season=2024&team=0&stats=sta
&pageitems=2000000000&v_cr=202301&qual=20"
# Initialize the Edge WebDriver
driver = webdriver.Edge()

# Open the webpage
driver.get(url)

# Find and click the "Sign In" button
sign_in_button = WebDriverWait(driver, 10).until(
    EC.element_to_be_clickable((By.XPATH, "//*[@id='navBar']/div[3]/ul/li[13]/
a/div/div[1]"))
)
sign_in_button.click()

# Wait for the sign-in form to appear
WebDriverWait(driver, 10).until(
    EC.visibility_of_element_located((By.ID, "user_login"))
)

# Locate and fill the username field
username_field = driver.find_element(By.ID, "user_login")
username_field.send_keys("aflynn0213")

# Locate and fill the password field
password_field = driver.find_element(By.ID, "user_pass")
password_field.send_keys("funfunfun123!")

# Submit the sign-in form
submit_button = driver.find_element(By.ID, "wp-submit")
submit_button.click()

# Wait for the export data button to appear and be clickable
export_button = WebDriverWait(driver, 30).until(
    EC.element_to_be_clickable((By.XPATH, "//*[@id='content']/div[16]/a"))
)

# Scroll to the export data button and click it using JavaScript
driver.execute_script("arguments[0].scrollIntoView(true);", export_button)
driver.execute_script("arguments[0].click();", export_button)

# Sleep for a while to ensure the download is initiated
time.sleep(10)
```

```
# Path to the directory where the file is downloaded
download_directory = "C:/Users/aflyn/Downloads/"

# Path to the directory where you want to move the downloaded file
download_path = "C:/Users/aflyn/repos/FantasyPlayerEvaluation/Expected_K_BB%/"

# Wait for the file to be downloaded
timeout = 30  # Adjust timeout as needed
start_time = time.time()
while not any(fname.startswith("fangraphs-leaderboards") for fname in os.listd
ir(download_directory)):
    if time.time() - start_time > timeout:
        print("Timeout occurred while waiting for the file to be downloaded.")
        break

# Move the downloaded file to the desired location
downloaded_files = [fname for fname in os.listdir(download_directory) if fnam
e.startswith("fangraphs-leaderboards")]
if downloaded_files:
    # Sort files based on modification time to get the latest one
    downloaded_files.sort(key=lambda x: os.path.getmtime(os.path.join(download
_directory, x)), reverse=True)

    # Assuming the first file in the sorted list is the latest one
    downloaded_file = downloaded_files[0]

    source_path = os.path.join(download_directory, downloaded_file)
    destination_path = os.path.join(download_path, "test.csv")  # Save as "tes
t.csv"
    shutil.move(source_path, destination_path)
    print(f"Latest file '{downloaded_file}' moved to '{destination_path}'.")
else:
    print("No file starting with 'fangraphs-leaderboards' found.")

# Close the WebDriver
driver.quit()
'''
```

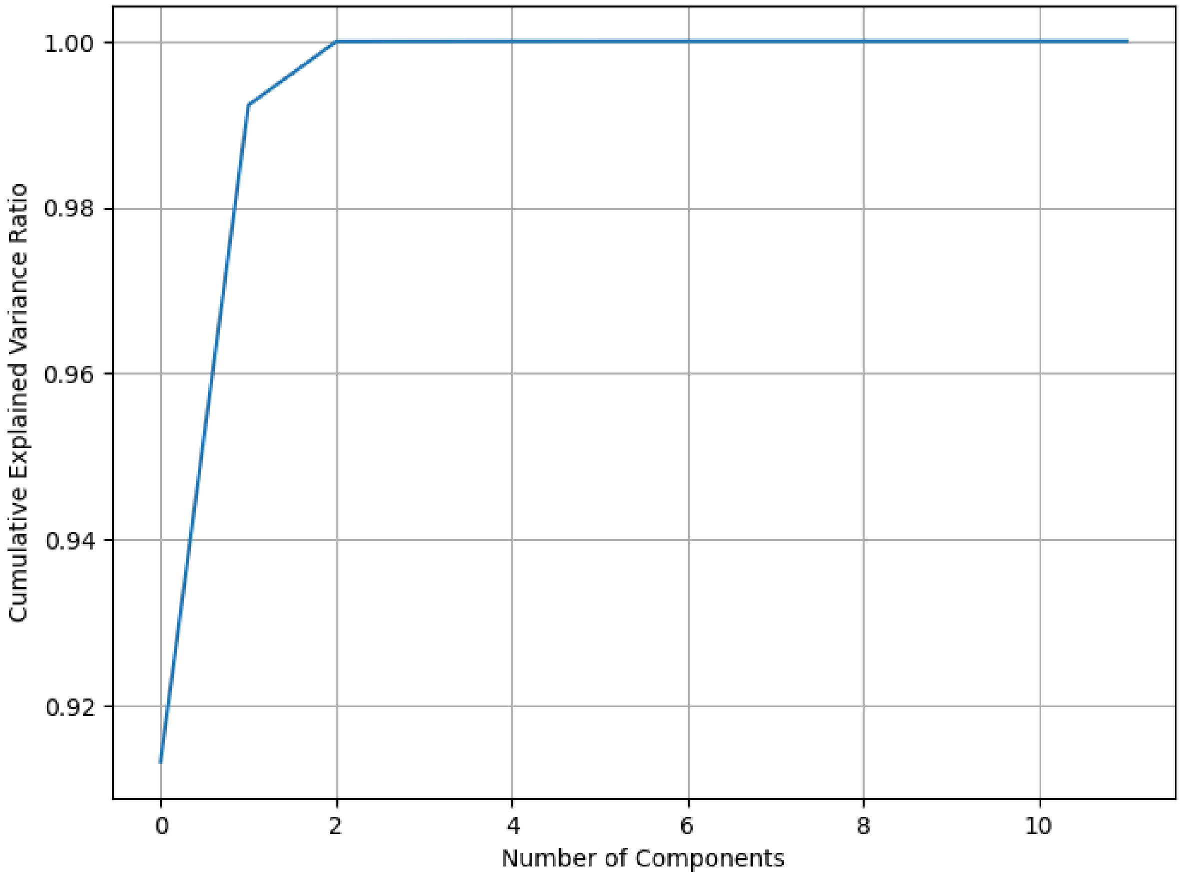
Out[78]:

```
'from selenium import webdriver\nfrom selenium.webdriver.common.by import By
\nfrom selenium.webdriver.support.ui import WebDriverWait\nfrom selenium.webd
river.support import expected_conditions as EC\nimport time\nimport shutil\ni
mport os\n\n# URL of the website\nurl = "https://www.fangraphs.com/leaders/ma
jor-league?pos=all&lg=all&type=c%2C13%2C6%2C38%2C41%2C42%2C43%2C44%2C45%2C47%
2C48%2C49%2C50%2C51%2C62%2C113%2C120%2C121%2C217%2C122%2C124%2C165%2C325%2C32
8%2C332%2C368%2C386%2C387%2C388%2C105%2C106%2C107%2C108%2C109%2C110%2C111%2C3
30%2C331&month=0&ind=1&rost=0&age=0&filter=&players=0&startdate=&enddate=&sea
son1=2024&season=2024&team=0&stats=sta&pageitems=2000000000&v_cr=202301&qual=
20"\n# Initialize the Edge WebDriver\ndriver = webdriver.Edge()\n\n# Open the
webpage\ndriver.get(url)\n\n# Find and click the "Sign In" button\nsign_in_bu
tton = WebDriverWait(driver, 10).until(\n    EC.element_to_be_clickable((By.X
PATH, "//*[@id=\'navBar\']/div[3]/ul/li[13]/a/div/div[1]"))\n)\nsign_in_butto
n.click()\n\n# Wait for the sign-in form to appear\nWebDriverWait(driver, 1
0).until(\n    EC.visibility_of_element_located((By.ID, "user_login"))\n)\n\n
# Locate and fill the username field\nusername_field = driver.find_element(B
y.ID, "user_login")\nusername_field.send_keys("aflynn0213")\n\n# Locate and f
ill the password field\npassword_field = driver.find_element(By.ID, "user_pas
s")\npassword_field.send_keys("funfunfun123!")\n\n# Submit the sign-in form\n
submit_button = driver.find_element(By.ID, "wp-submit")\nsubmit_button.click
()\n\n# Wait for the export data button to appear and be clickable\nexport_bu
tton = WebDriverWait(driver, 30).until(\n    EC.element_to_be_clickable((By.X
PATH, "//*[@id=\'content\']/div[16]/a"))\n)\n\n# Scroll to the export data bu
tton and click it using JavaScript\ndriver.execute_script("arguments[0].scrol
lIntoView(true);", export_button)\ndriver.execute_script("arguments[0].click
();", export_button)\n\n# Sleep for a while to ensure the download is initiat
ed\ntime.sleep(10)\n\n# Path to the directory where the file is downloaded\nd
ownload_directory = "C:/Users/aflyn/Downloads/"\n\n# Path to the directory wh
ere you want to move the downloaded file\ndownload_path = "C:/Users/aflyn/rep
os/FantasyPlayerEvaluation/Expected_K_BB%/"\n\n# Wait for the file to be down
loaded\ntimeout = 30  # Adjust timeout as needed\nstart_time = time.time()\nw
hile not any(fname.startswith("fangraphs-leaderboards") for fname in os.listd
ir(download_directory)):\n    if time.time() - start_time > timeout:\n
print("Timeout occurred while waiting for the file to be downloaded.")\n
break\n\n# Move the downloaded file to the desired location\ndownloaded_files
= [fname for fname in os.listdir(download_directory) if fname.startswith("fan
graphs-leaderboards")]\nif downloaded_files:\n    # Sort files based on modif
ication time to get the latest one\n    downloaded_files.sort(key=lambda x: o
s.path.getmtime(os.path.join(download_directory, x)), reverse=True)\n    \n
# Assuming the first file in the sorted list is the latest one\n    downloade
d_file = downloaded_files[0]\n    \n    source_path = os.path.join(download_d
irectory, downloaded_file)\n    destination_path = os.path.join(download_pat
h, "test.csv")  # Save as "test.csv"\n    shutil.move(source_path, destinatio
n_path)\n    print(f"Latest file \'{downloaded_file}\' moved to \'{destinatio
n_path}\'.")\nelse:\n    print("No file starting with \'fangraphs-leaderboard
s\' found.")\n\n# Close the WebDriver\ndriver.quit()\n'
```

In [79]:
```
df = pd.read_csv("training.csv")
df = split_data(df)
x = pd.concat([df['swstr%'], df.loc[:,'stuff+':'cstr%']],axis=1)
k = df['k%']
bb= df['bb%']
print(x.columns)
print(x.columns[x.isna().any()].tolist())
print(x.shape)
#std_sc = StandardScaler().fit(X)
#X = std_sc.transform(X)
```

```
Empty DataFrame
Columns: [team, ip, era, k/bb, avg, whip, babip, fip, ld%, gb%, fb%, iffb%, h
r/fb, xfip, swstr%, k%, bb%, siera, e-f, fa-z (sc), k-bb%, barrel%, hardhit%,
xera, stf+ fa, stuff+, location+, pitching+, o-swing%, z-swing%, swing%, o-co
ntact%, z-contact%, contact%, zone%, cstr%, csw%, nameascii, playerid, mlbami
d]
Index: []

[0 rows x 40 columns]
Index(['swstr%', 'stuff+', 'location+', 'pitching+', 'o-swing%', 'z-swing%',
       'swing%', 'o-contact%', 'z-contact%', 'contact%', 'zone%', 'cstr%'],
      dtype='object')
[]
(497, 12)
```

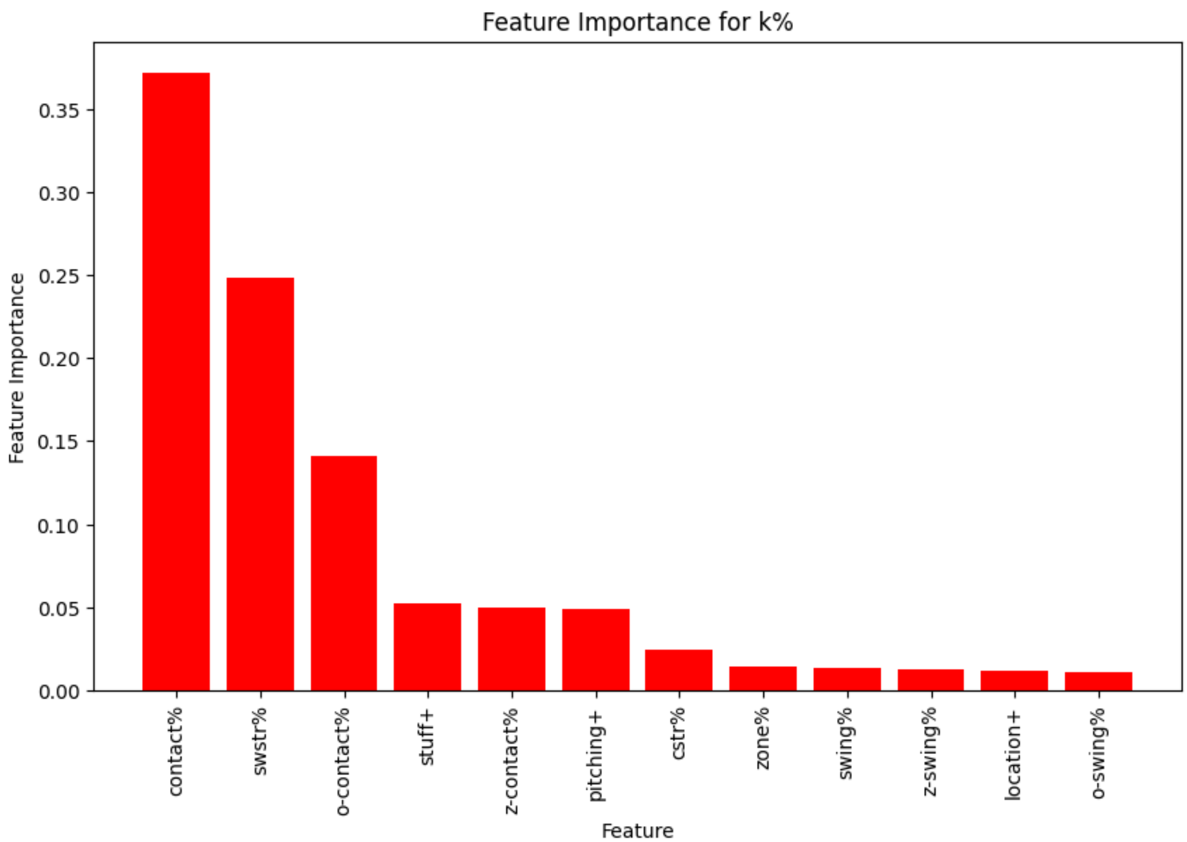
In [80]:
```
# Init PCA object
pca = PCA()
# Fit the PCA to your data
pca.fit(x)
# Plot cumulative explained variance ratio
plt.figure(figsize=(8, 6))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.grid(True)
plt.show()
```



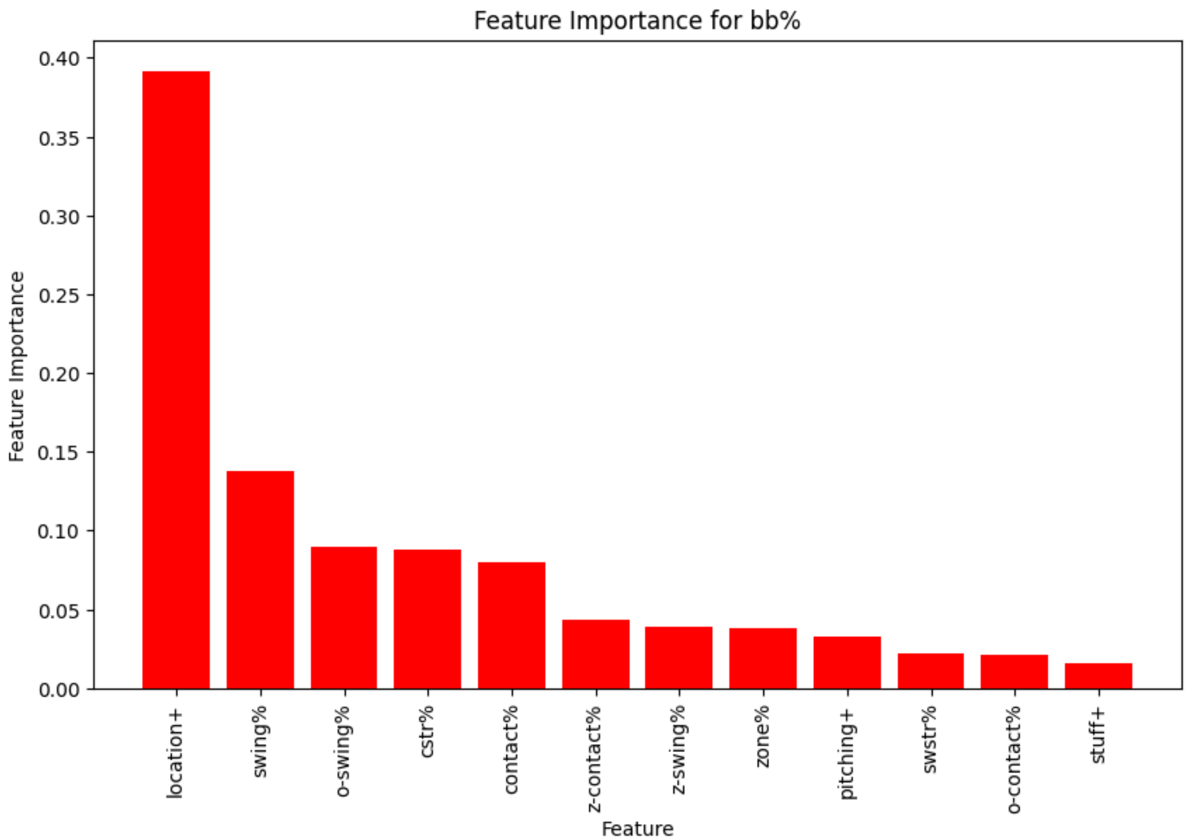
In [81]:
```
# Dictionary to hold models for each target variable
models = {}
################################ XGB REGRESSION CODE ########################
#####################################
# Initialize XGBoost regressor
xgb_regressor = xgb.XGBRegressor(objective='reg:squarederror')

# Define parameter grid
param_grid = {
    'max_depth': [3, 6, 9],  # Maximum depth of each tree
    'learning_rate': [0.01, 0.1, 0.3],  # Learning rate
    'n_estimators': [50, 100, 200],  # Number of boosting rounds
    'colsample_bytree': [0.6, 0.8, 1.0],  # Subsample ratio of columns when co
nstructing each tree
    'gamma': [0, 0.1, 0.3],  # Minimum loss reduction required to make a furth
er partition on a leaf node of the tree
    'reg_alpha': [0, 0.1, 0.3],  # L1 regularization term on weights
    'reg_lambda': [0, 0.1, 0.3]  # L2 regularization term on weights
}


# List of target variable column names
target_columns = [k.name, bb.name]
feature_names = x.columns.tolist()
# Iterate over target columns
for target_column in target_columns:
    # Initialize GridSearchCV
    model_gridsearch = GridSearchCV(
        estimator=xgb_regressor,
        param_grid=param_grid,
        scoring='neg_mean_squared_error',
        cv=5,  # 5-fold cross-validation
        n_jobs=-1  # Use all available CPU cores
    )

    # Fit the grid search to the data for the current target column
    model_gridsearch.fit(x, df[target_column])

    # Store the best model in the dictionary
    models[target_column+'_xgb'] = model_gridsearch.best_estimator_

    # Print the best parameters found
    print(f"Best Parameters for {target_column}:", model_gridsearch.best_param
s_)
    # Get feature importances for the best model
    best_model = models[target_column+'_xgb']
    feats_value = best_model.feature_importances_

    # Get the indices of features sorted by importance
    feats = np.argsort(feats_value)[::-1]

    print(f"Feature ranking for {target_column}:")
    for i, idx in enumerate(feats):
        print(f"{i + 1}. Feature {feature_names[idx]}: {feats_value[idx]}")

    # Plot feature importance for the best model
```

```
    plt.figure(figsize=(10, 6))
    plt.title(f"Feature Importance for {target_column}")
    plt.bar(range(len(feature_names)), feats_value[feats], color="r", align="center")
    plt.xticks(range(len(feature_names)), [feature_names[idx] for idx in feats], rotation=90)
    plt.xlabel("Feature")
    plt.ylabel("Feature Importance")
    plt.show()
###########################################################################################################
```

```
Best Parameters for k%: {'colsample_bytree': 0.6, 'gamma': 0, 'learning_rat
e': 0.1, 'max_depth': 3, 'n_estimators': 100, 'reg_alpha': 0, 'reg_lambda':
0}
Feature ranking for k%:
1. Feature contact%: 0.3720402717590332
2. Feature swstr%: 0.24852590262889862
3. Feature o-contact%: 0.14124412834644318
4. Feature stuff+: 0.05210644379258156
5. Feature z-contact%: 0.04955068975687027
6. Feature pitching+: 0.048524681478738785
7. Feature cstr%: 0.024645153433084488
8. Feature zone%: 0.014303945936262608
9. Feature swing%: 0.013595142401754856
10. Feature z-swing%: 0.012716786935925484
11. Feature location+: 0.011981751769781113
12. Feature o-swing%: 0.010765031911432743
```



```
Best Parameters for bb%: {'colsample_bytree': 0.6, 'gamma': 0, 'learning_rat
e': 0.1, 'max_depth': 3, 'n_estimators': 50, 'reg_alpha': 0, 'reg_lambda': 0.
3}
Feature ranking for bb%:
1. Feature location+: 0.3915686309337616
2. Feature swing%: 0.13775786757469177
3. Feature o-swing%: 0.09016229957342148
4. Feature cstr%: 0.08758064359426498
5. Feature contact%: 0.07976207137107849
6. Feature z-contact%: 0.043096158653497696
7. Feature z-swing%: 0.03867427632212639
8. Feature zone%: 0.0385248027741909
9. Feature pitching+: 0.03309711068868637
10. Feature swstr%: 0.02248617261648178
11. Feature o-contact%: 0.021487323567271233
12. Feature stuff+: 0.015802616253495216
```



In [82]:

```python
# Fit models for both 'k%' and 'bb%' target variables
models['k%_poly'] = polynomial_learning(x, k, k.name)
models['bb%_poly'] = polynomial_learning(x, bb, bb.name)
```

```
Best regressor for k%: Ridge
BEST SCORE for k%: 0.00045132651529260654
Coefficients: [ 0.00000000e+00  2.48835444e-01  3.76882638e-04  8.97753967e-0
4
  1.73922197e-03 -3.22489665e-02 -4.20339541e-02 -5.21468586e-02
 -1.10924156e-01 -3.83210336e-01 -3.83176099e-01  4.56621331e-04
  2.62514645e-01]
Best training parameters for k%:  {'alpha': 0.01, 'copy_X': True, 'fit_interc
ept': True, 'max_iter': None, 'positive': False, 'random_state': None, 'solve
r': 'auto', 'tol': 0.0001}
Best regressor for bb%: Ridge
BEST SCORE for bb%: 0.00011993059706380023
Coefficients: [ 0.00000000e+00 -6.19466946e-05 -9.72867173e-04 -8.62844011e-0
3
 -4.14116235e-03 -1.74667141e-04 -6.91429514e-05 -1.54871115e-04
 -1.50165808e-05  3.26964940e-05 -9.65453173e-06 -4.39105403e-05
  5.58333691e-05  7.86155944e-06 -6.46394287e-04 -5.49568839e-03
 -2.14861094e-03 -2.26160780e-05  3.45795656e-06 -1.18611147e-05
 -1.23093491e-04 -2.27349952e-05 -6.38736719e-05 -2.54244485e-05
 -2.68332860e-05 -1.34836389e-05 -5.32684453e-05  8.72952142e-05
 -2.45596675e-04 -1.38636467e-03 -1.29935332e-03  5.05588663e-04
  2.93589654e-03  5.00158456e-04 -1.06209230e-04 -6.22702584e-03
 -2.55077562e-05  2.22691512e-04 -1.69820261e-03 -5.01689304e-04
 -1.79413472e-03  1.03250209e-03 -2.17556244e-03 -1.89403033e-03
 -1.36558907e-03  1.23128718e-03 -1.22650328e-04  1.20751879e-03
  1.06793947e-03  9.73774962e-04 -1.49025740e-03 -1.71280397e-03
 -2.74629843e-03  1.49262703e-03 -1.00105959e-03 -2.58558598e-05
 -2.26972526e-07  2.07528625e-05 -9.62853853e-05 -1.05662068e-04
 -9.15500567e-05  2.57083022e-05 -4.96142721e-05 -1.17803844e-04
 -3.88472581e-05 -2.99299253e-04 -6.27812501e-06 -1.13838684e-04
  4.55045988e-05  8.27573149e-05  2.51950748e-05 -1.60135641e-04
 -8.48955395e-05 -1.02508910e-04  4.03042120e-05 -4.42555080e-06
 -7.14780489e-05  1.96639935e-04  1.03764311e-04  9.09905869e-05
  1.77684225e-04 -1.46438365e-04 -4.37548649e-05 -3.53948295e-05
  2.47276237e-05  8.25270100e-06  2.08146571e-05  8.59721343e-05
  2.61778693e-05 -2.60519036e-06 -1.33294850e-05]
Best training parameters for bb%:  {'alpha': 2.5, 'copy_X': True, 'fit_interc
ept': True, 'max_iter': None, 'positive': False, 'random_state': None, 'solve
r': 'auto', 'tol': 0.0001}
```

In [84]:

```
df_test = pd.read_csv("test.csv")
df_test = split_data(df_test)
x_test = pd.concat([df_test['swstr%'], df_test.loc[:,'stuff+':'cstr%']],axis=
1)
k_test= df_test['k%']
bb_test = df_test['bb%']

predictions_df = pd.DataFrame(index=x_test.index)

for name,model in models.items():
    if 'k%' in name:
        test_data = k_test
    elif 'bb%' in name:
        test_data = bb_test
    else:
        continue  # Skip models not related to 'k%' or 'bb%'
    y_preds = evaluate_model(model, x_test, test_data, name)
    predictions_df[name] = y_preds

predictions_df["k%"] = k_test
predictions_df["bb%"] = bb_test
predictions_df = calculate_k_bb(predictions_df)
# Write the DataFrame to a CSV file
predictions_df.to_csv('pred_k_bb%.csv')
```

```
Empty DataFrame
Columns: [team, ip, era, k/bb, avg, whip, babip, lob%, fip, ld%, gb%, fb%, if
fb%, hr/fb, xfip, swstr%, k%, bb%, k-bb%, siera, e-f, fa-z (sc), barrel%, har
dhit%, xera, stf+ fa, stuff+, location+, pitching+, o-swing%, z-swing%, swin
g%, o-contact%, z-contact%, contact%, zone%, cstr%, csw%, nameascii, playeri
d, mlbamid]
Index: []

[0 rows x 41 columns]
XGB MODEL
Evaluation metrics for k%_xgb:
Model Type XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.6, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=0, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=3, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=100, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...):
Mean Squared Error (MSE): 0.0007623259714061682
Mean Absolute Error (MAE): 0.022287880927194074
R-squared (R2) Score: 0.6695073481151572

XGB MODEL
Evaluation metrics for bb%_xgb:
Model Type XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.6, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=0, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=3, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=50, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...):
Mean Squared Error (MSE): 0.00018429641674586392
Mean Absolute Error (MAE): 0.010505127291451394
R-squared (R2) Score: 0.6532123597519367

Evaluation metrics for k%_poly:
Model Type Ridge(alpha=0.01):
Mean Squared Error (MSE): 0.0006946455084237O8
Mean Absolute Error (MAE): 0.021232035103378333
R-squared (R2) Score: 0.6988489900516215

Evaluation metrics for bb%_poly:
Model Type Ridge(alpha=2.5):
Mean Squared Error (MSE): 0.0001751206776116751
Mean Absolute Error (MAE): 0.010624029367618292
R-squared (R2) Score: 0.6704782023443352

k%_xgb
bb%_xgb
```

k%_poly
bb%_poly
k%
NOT A MODEL OUTPUT COLUMN
bb%
NOT A MODEL OUTPUT COLUMN

In [85]:
```
players_2021 = df.index.get_level_values('name')[df.index.get_level_values('se
ason') == 2021].unique()
players_2022 = df.index.get_level_values('name')[df.index.get_level_values('se
ason') == 2022].unique()
common_players = set(players_2021).intersection(players_2022)


players_2023 = df.index.get_level_values('name')[df.index.get_level_values('se
ason') == 2023].unique()
common_players_2 = set(players_2022).intersection(players_2023)
print(common_players_2)
data_2021_x = df.loc[(2021, list(common_players)), :]
data_2022_y = df.loc[(2022, list(common_players)), :]
data_2022_x = df.loc[(2022, list(common_players_2)), :]
data_2023_y = df.loc[(2023, list(common_players_2)), :]
df_x = pd.concat([data_2021_x, data_2022_x])
df_y = pd.concat([data_2022_y, data_2023_y])

print(df_x,df_y)
x = pd.concat([df_x['swstr%'], df_x.loc[:,'stuff+':'cstr%']],axis=1)
k_yty = df_y['k%']
bb_yty = df_y['bb%']

k_x = models['k%_poly'].named_steps['poly'].transform(x)
preds_k = models['k%_poly'].named_steps['regressor'].predict(k_x)
preds_k_xgb = models['k%_xgb'].predict(x)
r2_kk = r2_score(k_yty,preds_k)
r2_kk_xgb = r2_score(k_yty,preds_k_xgb)

bb_x = models['bb%_poly'].named_steps['poly'].transform(x)
preds_bb = models['bb%_poly'].named_steps['regressor'].predict(bb_x)
preds_bb_xgb = models['bb%_xgb'].predict(x)
r2_bb = r2_score(bb_yty,preds_bb)
r2_bb_xgb = r2_score(bb_yty,preds_bb_xgb)

print("K% R2 YEAR TO YEAR: ", r2_score(k_yty,df_x['k%']))
print("K% Poly Model R2 YEAR TO YEAR: ", r2_kk)
print("K% XGB Model R2 YEAR TO YEAR: ", r2_kk_xgb)
print("BB% R2 YEAR TO YEAR: ", r2_score(bb_yty,df_x['bb%']))
print("BB% Poly Model R2 YEAR TO YEAR: ", r2_bb)
print("BB% XGB Model R2 YEAR TO YEAR: ", r2_kk)
```

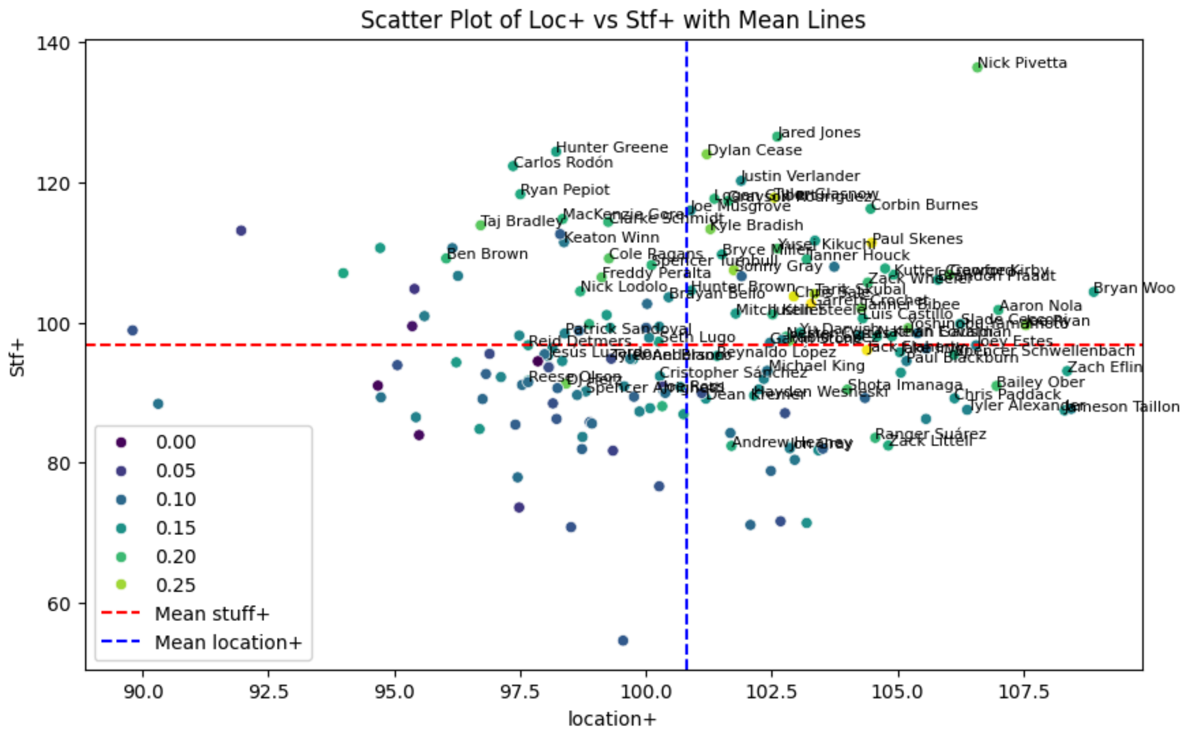
```
{'Spencer Strider', 'Michael Lorenzen', 'Tyler Wells', 'David Peterson', 'Bra
dy Singer', 'Bailey Falter', 'Lucas Giolito', 'Adam Wainwright', 'Jesús Luzar
do', 'Miles Mikolas', 'Dane Dunning', 'Framber Valdez', 'Patrick Sandoval',
'Justin Steele', 'Braxton Garrett', 'Sean Manaea', 'Nick Pivetta', 'Logan Web
b', 'Nick Martinez', 'Shohei Ohtani', 'Julio Urías', 'Marcus Stroman', 'Luis
Severino', 'Paul Blackburn', 'Reid Detmers', 'José Berríos', 'Shane McClanaha
n', 'Chris Flexen', 'Josiah Gray', 'Dylan Cease', 'Adrian Houser', 'Sandy Alc
antara', 'Sonny Gray', 'Shane Bieber', 'Alek Manoah', 'Jordan Lyles', 'Carlos
Carrasco', 'Austin Gomber', 'Ryan Yarbrough', 'Tarik Skubal', 'Alex Cobb', 'A
aron Nola', 'Chris Bassitt', 'Graham Ashcraft', 'Trevor Williams', 'Mitch Kel
ler', 'Kyle Bradish', 'Tyler Anderson', 'Jameson Taillon', 'Kyle Freeland',
'Clayton Kershaw', 'Max Scherzer', 'Jon Gray', 'Tony Gonsolin', 'Cristian Jav
ier', 'Yusei Kikuchi', 'Charlie Morton', 'Joe Musgrove', 'Pablo López', 'Nath
an Eovaldi', 'Rich Hill', 'Patrick Corbin', 'Cal Quantrill', 'Jordan Montgome
ry', 'Justin Verlander', 'Noah Syndergaard', 'Michael Wacha', 'Merrill Kell
y', 'Kyle Gibson', 'Zach Davies', 'Michael Kopech', 'Logan Gilbert', 'Zac Gal
len', 'Joe Ryan', 'Jakob Junis', 'Eduardo Rodriguez', 'Taijuan Walker', 'Corb
in Burnes', 'Blake Snell', 'Kyle Hendricks', 'Gerrit Cole', 'Alex Wood', 'Aar
on Civale', 'Zack Wheeler', 'Kevin Gausman', 'Dakota Hudson', 'Ross Striplin
g', 'Luis Castillo', 'Yu Darvish', 'George Kirby', 'Lance Lynn', 'Dean Kreme
r', 'Mike Clevinger', 'Martín Pérez', 'Hunter Greene', 'Zack Greinke', 'Range
r Suárez', 'Drew Smyly'}
                       team     ip       era      k/bb       avg      whip  \
season name
2021   Dylan Bundy      LAA   90.2  6.055147  2.470588  0.249300  1.356618
       Brady Singer     KCR  128.1  4.909091  2.471698  0.279693  1.550649
       Lucas Giolito    CHW  178.2  3.526120  3.865385  0.217718  1.102612
       Adam Wainwright  STL  206.1  3.053312  3.480000  0.218466  1.056543
       Jake Odorizzi    HOU  104.2  4.213376  2.676471  0.240099  1.251592
...                     ...    ...       ...       ...       ...       ...
2022   Martín Pérez     TEX  196.1  2.887946  2.449275  0.238606  1.258065
       Hunter Greene    CIN  125.2  4.440318  3.416667  0.219873  1.209549
       Zack Greinke     KCR  137.0  3.678832  2.703704  0.282883  1.343066
       Ranger Suárez    PHI  155.1  3.650215  2.224138  0.247508  1.332618
       Drew Smyly       CHC  106.1  3.470220  3.500000  0.242788  1.194357

                           babip       fip       ld%       gb%  ...    swing%
\
season name                                                     ...
2021   Dylan Bundy      0.272727  5.508270  0.207407  0.407407  ...  0.444152
       Brady Singer     0.350133  4.042761  0.223650  0.498715  ...  0.432444
       Lucas Giolito    0.269406  3.791303  0.245119  0.331887  ...  0.503367
       Adam Wainwright  0.256098  3.664380  0.222034  0.474576  ...  0.433507
       Jake Odorizzi    0.272727  4.478951  0.200647  0.352751  ...  0.489442
...                          ...       ...       ...       ...  ...       ...
2022   Martín Pérez     0.295053  3.265232  0.187943  0.514184  ...  0.442430
       Hunter Greene    0.280702  4.369725  0.221498  0.293160  ...  0.487273
       Zack Greinke     0.305556  4.032139  0.233333  0.412500  ...  0.460387
       Ranger Suárez    0.292576  3.865650  0.174840  0.554371  ...  0.453198
       Drew Smyly       0.275081  4.231553  0.175926  0.401235  ...  0.519294

                        o-contact%  z-contact%  contact%     zone%     cstr%
\
season name
2021   Dylan Bundy        0.616279    0.889952  0.785503  0.440867  0.204068
       Brady Singer       0.604905    0.858521  0.764408  0.434193  0.200700
       Lucas Giolito      0.545956    0.783386  0.696990  0.441751  0.145455
```

```
       Adam Wainwright    0.674858    0.903870  0.812782  0.431877  0.216428
       Jake Odorizzi      0.707386    0.865942  0.804204  0.402815  0.141852
...                            ...         ...       ...       ...       ...
2022   Martín Pérez       0.739933    0.868421  0.810319  0.359181  0.188318
       Hunter Greene      0.550971    0.796970  0.702425  0.413636  0.146818
       Zack Greinke       0.768889    0.897651  0.842256  0.398327  0.176937
       Ranger Suárez      0.688797    0.889706  0.806368  0.386115  0.167317
       Drew Smyly         0.627397    0.845754  0.761146  0.446527  0.150496

                              csw%        nameascii  playerid  mlbamid
season name
2021   Dylan Bundy      0.299213      Dylan Bundy     12917   605164
       Brady Singer     0.302580     Brady Singer     25377   663903
       Lucas Giolito    0.297980    Lucas Giolito     15474   608337
       Adam Wainwright  0.297588  Adam Wainwright      2233   425794
       Jake Odorizzi    0.237683    Jake Odorizzi      6397   543606
...                          ...              ...       ...      ...
2022   Martín Pérez     0.272239     Martin Perez      6902   527048
       Hunter Greene    0.291818    Hunter Greene     22182   668881
       Zack Greinke     0.249560     Zack Greinke      1943   425844
       Ranger Suárez    0.255070    Ranger Suarez     17277   624133
       Drew Smyly       0.274531       Drew Smyly     11760   592767

[205 rows x 40 columns]                         team     ip       era
k/bb       avg      whip  \
season name
2022   Dylan Bundy        MIN  140.0  4.885714  3.357143  0.267730  1.278571
       Brady Singer       KCR  153.1  3.228261  4.285714  0.243478  1.141304
       Lucas Giolito      CHW  161.2  4.898969  2.901639  0.270142  1.435052
       Adam Wainwright    STL  191.2  3.709565  2.648148  0.258760  1.283478
       Jake Odorizzi    ---  106.1  4.401254  2.457143  0.255422  1.326019
...                       ...    ...       ...       ...       ...       ...
2023   Martín Pérez       TEX  141.2  4.447059  1.897959  0.270270  1.404706
       Hunter Greene      CIN  112.0  4.821429  3.166667  0.251131  1.419643
       Zack Greinke       KCR  142.1  5.058548  4.217391  0.279152  1.271663
       Ranger Suárez      PHI  125.0  4.176000  2.479167  0.263265  1.416000
       Drew Smyly         CHC  142.1  4.995317  2.517857  0.260638  1.426230

                           babip       fip       ld%       gb%  ...    swing%
\
season name                                                     ...
2022   Dylan Bundy      0.284753  4.662431  0.201285  0.340471  ...  0.516514
       Brady Singer     0.299754  3.581996  0.200957  0.490431  ...  0.441919
       Lucas Giolito    0.340278  4.058823  0.246696  0.385463  ...  0.472939
       Adam Wainwright  0.301887  3.660257  0.238731  0.432387  ...  0.434089
       Jake Odorizzi    0.292063  4.278575  0.216463  0.317073  ...  0.500000
...                          ...       ...       ...       ...  ...       ...
2023   Martín Pérez     0.292517  4.991511  0.189011  0.454945  ...  0.449843
       Hunter Greene    0.339483  4.246112  0.189474  0.343860  ...  0.489943
       Zack Greinke     0.299550  4.744502  0.206009  0.431330  ...  0.465525
       Ranger Suárez    0.324022  3.903040  0.207084  0.485014  ...  0.447355
       Drew Smyly       0.304786  4.955275  0.213429  0.347722  ...  0.465223

                        o-contact%  z-contact%  contact%     zone%     cstr%
\
season name
2022   Dylan Bundy        0.695946    0.887097  0.811723  0.456422  0.169266
```

```
       Brady Singer       0.630208    0.881381  0.789524  0.452862  0.210438
       Lucas Giolito      0.565966    0.861361  0.742704  0.409008  0.168907
       Adam Wainwright    0.741935    0.909964  0.844853  0.420683  0.211299
       Jake Odorizzi      0.710145    0.848375  0.795328  0.398220  0.134038
...                            ...         ...       ...       ...       ...
2023   Martín Pérez       0.743341    0.902896  0.837000  0.398560  0.184636
       Hunter Greene      0.581683    0.819063  0.725318  0.420498  0.145246
       Zack Greinke       0.735802    0.920382  0.848015  0.425417  0.175146
       Ranger Suárez      0.657431    0.893701  0.790055  0.392981  0.178360
       Drew Smyly         0.633803    0.851613  0.754700  0.382341  0.164173

                              csw%        nameascii  playerid  mlbamid
season name
2022   Dylan Bundy      0.266514      Dylan Bundy     12917   605164
       Brady Singer     0.303451     Brady Singer     25377   663903
       Lucas Giolito    0.290592    Lucas Giolito     15474   608337
       Adam Wainwright  0.278647  Adam Wainwright      2233   425794
       Jake Odorizzi    0.236374    Jake Odorizzi      6397   543606
...                          ...              ...       ...      ...
2023   Martín Pérez     0.257862     Martin Perez      6902   527048
       Hunter Greene    0.279503    Hunter Greene     22182   668881
       Zack Greinke     0.245835     Zack Greinke      1943   425844
       Ranger Suárez    0.272233    Ranger Suarez     17277   624133
       Drew Smyly       0.278055       Drew Smyly     11760   592767

[205 rows x 40 columns]
K% R2 YEAR TO YEAR:  0.4850678425017747
K% Poly Model R2 YEAR TO YEAR:  0.4374246669785169
K% XGB Model R2 YEAR TO YEAR:  0.5108300330401269
BB% R2 YEAR TO YEAR:  0.021924295854467535
BB% Poly Model R2 YEAR TO YEAR:  0.09642719162266844
BB% XGB Model R2 YEAR TO YEAR:  0.4374246669785169
```

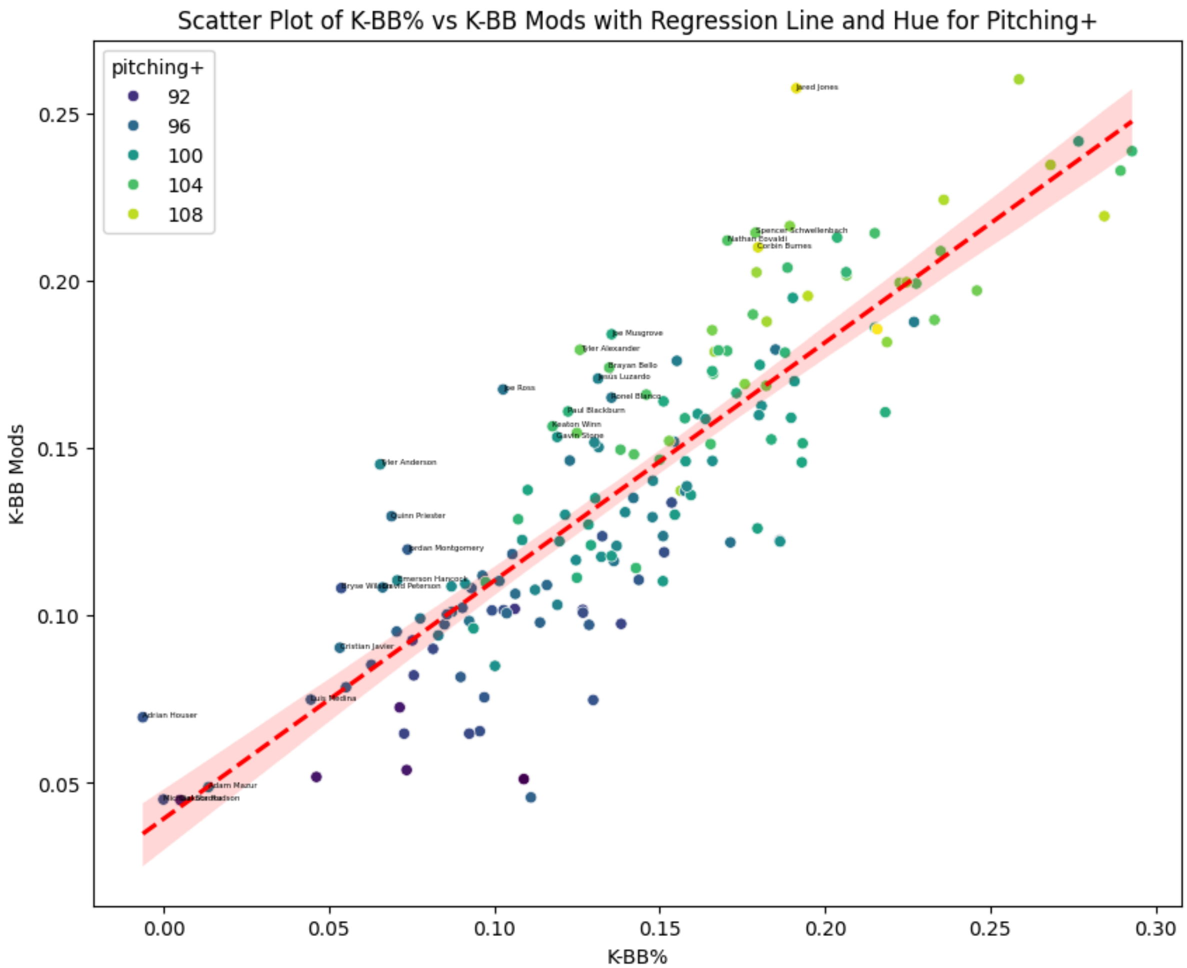
In [86]:

```
mean_1 = df_test['stuff+'].mean()
mean_2 = df_test['location+'].mean()
# Plotting
plt.figure(figsize=(10, 6))

# Scatter plot with color intensity based on another column and player names n
ear the dots
sns.scatterplot(x='location+', y='stuff+', hue='k-bb%', data=df_test, palette
='viridis')

# Plotting mean lines
plt.axhline(mean_1, color='red', linestyle='--', label='Mean stuff+')
plt.axvline(mean_2, color='blue', linestyle='--', label='Mean location+')

# Set labels and title
plt.xlabel('location+')
plt.ylabel('Stf+')
plt.title('Scatter Plot of Loc+ vs Stf+ with Mean Lines')

# Add legend
plt.legend()

# Filter rows where 'k-bb mods' is greater than 12 in predictions_df
filtered_rows = predictions_df[predictions_df['k-bb mods'] > predictions_df['k
-bb mods'].mean()]

# Show player names above mean lines
for i, row in df_test.iterrows():
    index = (i[0], i[1])
    if index in filtered_rows.index:
        player_name = filtered_rows.loc[index, :].name  # Retrieve player name
from filtered rows
        plt.text(row['location+'], row['stuff+'], player_name[1], fontsize=8,
color='black', ha='left')

plt.show()
```



Scatter Plot of Loc+ vs Stf+ with Mean Lines
Stf+
location+
140
120
100
80
60
90.0
92.5
95.0
97.5
100.0
102.5
105.0
107.5
0.00
0.05
0.10
0.15
0.20
0.25
Mean stuff+
Mean location+
Nick Pivetta
Jared Jones
Hunter Greene
Dylan Cease
Carlos Rodón
Justin Verlander
Ryan Pepiot
Corbin Burnes
Joe Musgrove
Taj Bradley
Kyle Bradish
Keaton Winn
Paul Skenes
Ben Brown
Cole Ragans
Tanner Houck
Sonny Gray
Freddy Peralta
Nick Lodolo
Hunter Brown
Bryan Woo
Aaron Nola
Patrick Sandoval
Seth Lugo
Reid Detmers
Reynaldo López
Michael King
Zach Eflin
Cristopher Sánchez
Shota Imanaga
Bailey Ober
Chris Paddack
Tyler Alexander
Jameson Taillon
Ranger Suárez
Zack Littell
Paul Blackburn
Joey Estes
Luis Castillo

In [87]:
```
plt.figure(figsize=(10, 8))  # Adjust the size as needed

# Sort DataFrame based on the difference between 'k-bb mods' and 'k-bb%'
sorted_df = predictions_df.assign(diff=lambda x: x['k-bb mods'] - df_test['k-b
b%']).sort_values(by='diff', ascending=False)

# Select the top 25 rows
top_25 = sorted_df.head(25)

# Plot scatter plot with hue
scatter = sns.scatterplot(x=df_test['k-bb%'], y=predictions_df['k-bb mods'], h
ue=df_test['pitching+'], palette='viridis')

# Overlay regression line y=x
sns.regplot(x=df_test['k-bb%'], y=predictions_df['k-bb mods'], scatter=False,
line_kws={'color': 'red', 'linestyle': '--'})

# Display names of top 25
for name, _ in top_25.iterrows():
    plt.text(df_test.loc[name, 'k-bb%'], predictions_df.loc[name, 'k-bb mod
s'], name[1], fontsize=4, color='black')

# Add labels and title
plt.xlabel('K-BB%')
plt.ylabel('K-BB Mods')
plt.title('Scatter Plot of K-BB% vs K-BB Mods with Regression Line and Hue for
Pitching+')
# Show plot
plt.show()
```



In [88]:

```
from sklearn.preprocessing import StandardScaler

# Prepare feature matrix X and target variable y
quick_cols = ['swstr%', 'contact%', 'o-contact%', 'stuff+','pitching+']
x_ks = df[quick_cols]
scaler = StandardScaler()
x_ks_scaled = scaler.fit_transform(x_ks)
y = df['k%']

quick_mod = polynomial_learning(x_ks_scaled,y,y.name)

x_test_ks = quick_mod["poly"].transform(scaler.transform(df_test[quick_cols]))
preds_ks = quick_mod["regressor"].predict(x_test_ks)

# Make predictions on the standardized test data
preds_ks = quick_mod["regressor"].predict(x_test_ks)
```

```
Best regressor for k%: Ridge
BEST SCORE for k%: 0.0005007261465317277
Coefficients: [ 0.         -0.00804518 -0.04578432  0.004417    0.00296472
0.01382614]
Best training parameters for k%:  {'alpha': 0.001, 'copy_X': True, 'fit_inter
cept': True, 'max_iter': None, 'positive': False, 'random_state': None, 'solv
er': 'auto', 'tol': 0.0001}
```

In [89]:

```
# Compute regression metrics
mae = mean_absolute_error(df_test['k%'], preds_ks)
mse = mean_squared_error(df_test['k%'], preds_ks)
rmse = np.sqrt(mse)
r2 = r2_score(df_test['k%'], preds_ks)

print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R-squared of simplified model:", r2)

if r2 < 0.7:
    print("My Larger Poly K% model WINS!")
else:
    print("Turns out simpler is better")
```

```
Mean Absolute Error: 0.022305699666583127
Mean Squared Error: 0.0007608617637367583
Root Mean Squared Error: 0.027583722804160396
R-squared of simplified model: 0.6701421288962458
My Larger Poly K% model WINS!
```

In [90]:

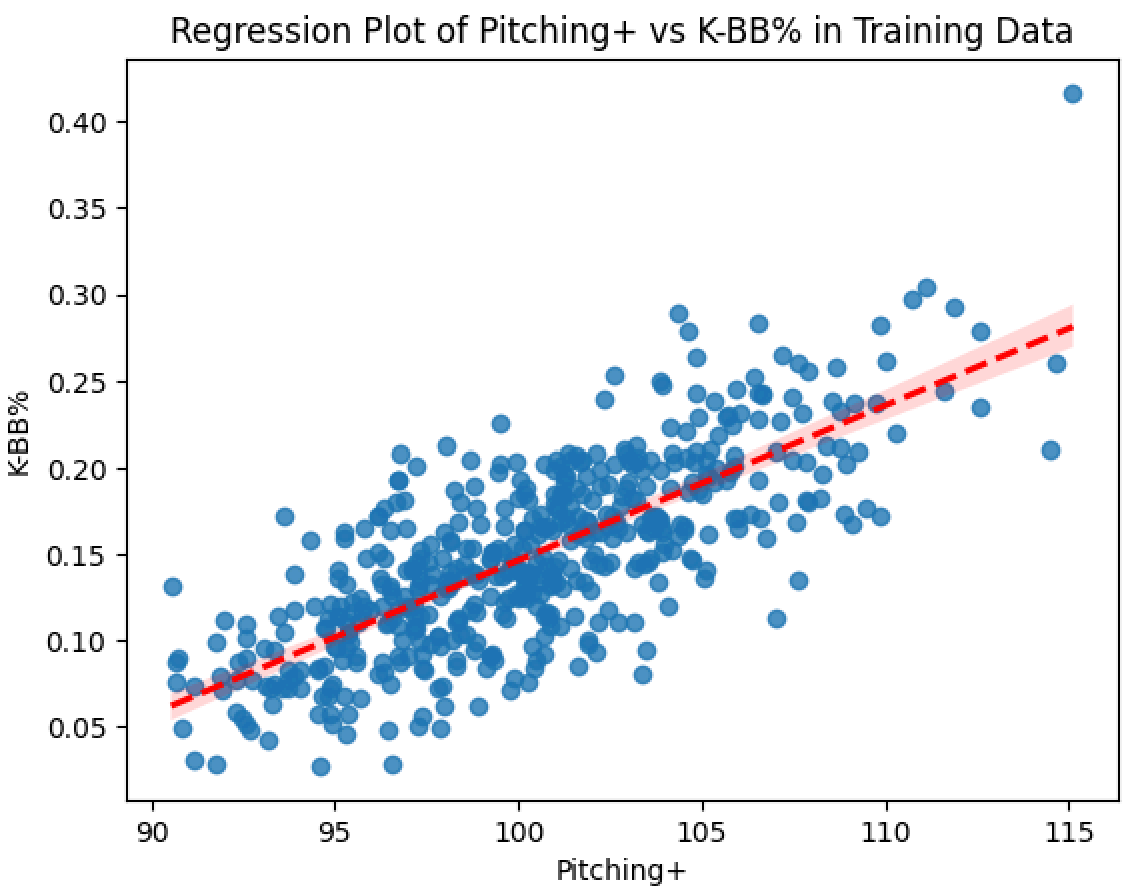
```
# Prepare feature matrix X and target variable y
quick_bb_cols = ['location+', 'o-swing%']  # Update to correct column names
x_bb = df[quick_bb_cols]
scaler = StandardScaler()
x_bb_scaled = scaler.fit_transform(x_bb)
y = df['bb%']

quick_bb = polynomial_learning(x_bb_scaled, y, y.name)

# Transform the test data
x_test_bb_scaled = scaler.transform(df_test[quick_bb_cols])  # Scale the test
data
x_test_bb_poly = quick_bb["poly"].transform(x_test_bb_scaled)  # Apply polynom
ial transformation
preds_bb = quick_bb["regressor"].predict(x_test_bb_poly)  # Make predictions
```

```
Best regressor for bb%: Ridge
BEST SCORE for bb%: 0.00016336858975364165
Coefficients: [ 0.         -0.01377837 -0.00344749  0.00130963  0.00088709 -
0.00107693]
Best training parameters for bb%:  {'alpha': 5, 'copy_X': True, 'fit_intercep
t': True, 'max_iter': None, 'positive': False, 'random_state': None, 'solve
r': 'auto', 'tol': 0.0001}
```

In [91]:
```
# Compute regression metrics
mae = mean_absolute_error(df_test['bb%'], preds_bb)
mse = mean_squared_error(df_test['bb%'], preds_bb)
rmse = np.sqrt(mse)
r2_bb = r2_score(df_test['bb%'], preds_bb)

print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R-squared of simplified model:", r2_bb)

if r2_bb < 0.66:
    print("My Larger Poly BB% model WINS!")
else:
    print("Turns out simpler is better")
```

```
Mean Absolute Error: 0.011382411894215854
Mean Squared Error: 0.00020231466100564964
Root Mean Squared Error: 0.014223735831547549
R-squared of simplified model: 0.619307715708419
My Larger Poly BB% model WINS!
```

In [92]:
```python
# Plot regression plot
sns.regplot(x=df['pitching+'], y=df['k-bb%'], line_kws={'color': 'red', 'linestyle': '--'})

# Add labels and title
plt.ylabel('K-BB%')
plt.xlabel('Pitching+')
plt.title('Regression Plot of Pitching+ vs K-BB% in Training Data')

# Show plot
plt.show()

from scipy.stats import linregress

# Perform linear regression
slope, intercept, r_value, p_value, std_err = linregress(df['pitching+'],df['k-bb%'])

# Print slope and correlation coefficient
print("Slope:", slope)
print("Correlation coefficient (r):", r_value)
print("P-value: ", p_value)
```




```
Slope: 0.008913488207496476
Correlation coefficient (r): 0.7534268496681034
P-value:  3.511474478773595e-92
```

In [93]:
```
pitch_preds = df_test['pitching+']*slope + intercept
print("R2 SCORE Pitching+ K-BB% : ", r2_score(df_test["k-bb%"],pitch_preds))
```

```
R2 SCORE Pitching+ K-BB% :  0.4758362844294972
```

In [ ]: