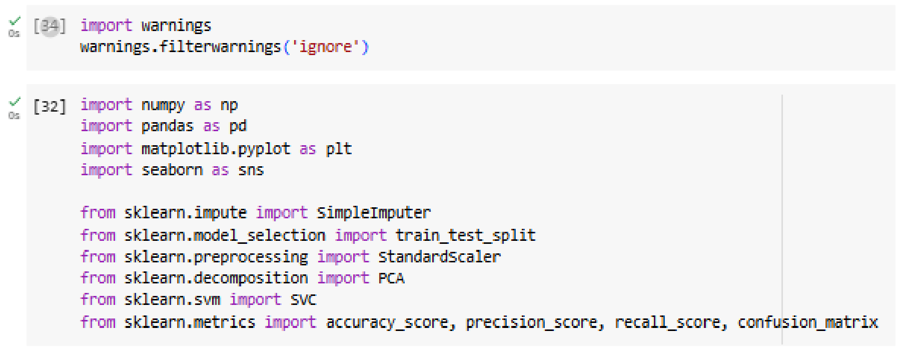
Name: Thanh Nguyen
Student ID: 801376954
Assignment #: 4

In this assignment, we will work with two datasets: the Cancer dataset and the Housing dataset. The objective is to apply machine learning techniques such as logistic regression, Naive Bayes classification, and support vector regression (SVR) to classify or predict outcomes based on the given data. You are encouraged to use built-in machine learning functions from libraries like scikit-learn for tasks including gradient descent, model training, and evaluation. The Cancer dataset will be used for classification tasks (malignant vs. benign), while the Housing dataset will be used for regression (predicting house prices). For both datasets, you must split the data into 80% for training and 20% for testing. Be sure to preprocess the data properly by handling missing values and encoding categorical variables where necessary. Sample code for accessing and preparing the datasets has been provided via Canvas to support your implementation.

Link on Github Repos: Assignment 4 GitHub Repos

Problem 1:

Use the cancer dataset to build an SVM classifier to classify the type of cancer (Malignant vs. benign). Use the PCA feature extraction for your training. Perform N number of independent training (N=1, …, K).

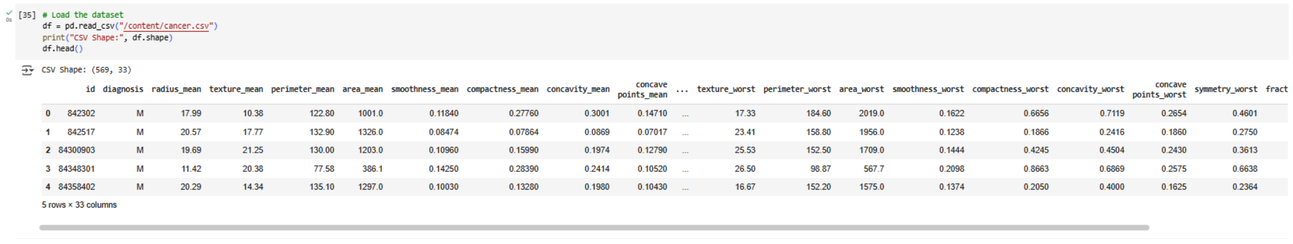


```
[34] import warnings
     warnings.filterwarnings('ignore')
```

```
[32] import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns

     from sklearn.impute import SimpleImputer
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
     from sklearn.decomposition import PCA
     from sklearn.svm import SVC
     from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix
```

In this section, we import all the necessary Python libraries for our machine learning project. I bring in tools for data handling (numpy, pandas), visualization (matplotlib.pyplot, seaborn), and machine learning (sklearn modules like train_test_split, StandardScaler, PCA, and SVC). I also import metrics like accuracy, precision, recall, and the confusion matrix to evaluate our models. Additionally, we suppress unnecessary warning messages with warnings.filterwarnings('ignore') to keep the output clean.

Name: Thanh Nguyen
Student ID: 801376954
Assignment #: 4



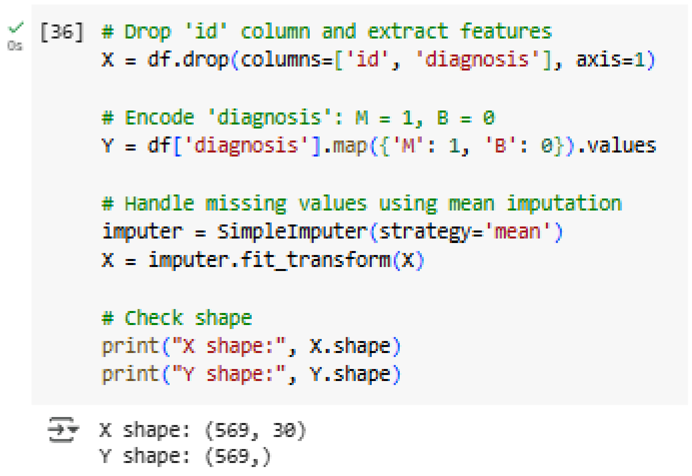
```
[35] # Load the dataset
     df = pd.read_csv("/content/cancer.csv")
     print("CSV Shape:", df.shape)
     df.head()
```

```
CSV Shape: (569, 33)
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	fract
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	

5 rows × 33 columns

Here, we read the cancer dataset (a CSV file) into a pandas DataFrame using pd.read_csv(). We then display its shape (569 rows and 33 columns) and preview the first few rows with df.head(). This step helps us understand what the data looks like, I see features like radius_mean, texture_mean, and more, along with diagnosis, which is the column we'll use as the label for classification (M = malignant, B = benign).

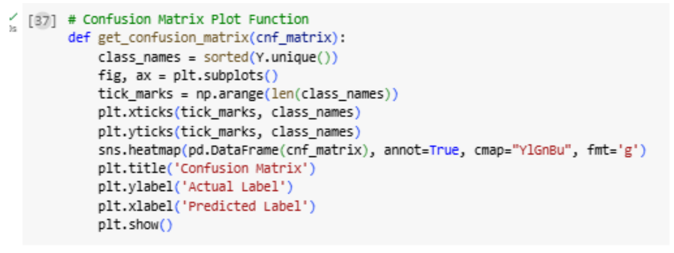


```
[36] # Drop 'id' column and extract features
     X = df.drop(columns=['id', 'diagnosis'], axis=1)

     # Encode 'diagnosis': M = 1, B = 0
     Y = df['diagnosis'].map({'M': 1, 'B': 0}).values

     # Handle missing values using mean imputation
     imputer = SimpleImputer(strategy='mean')
     X = imputer.fit_transform(X)

     # Check shape
     print("X shape:", X.shape)
     print("Y shape:", Y.shape)
```

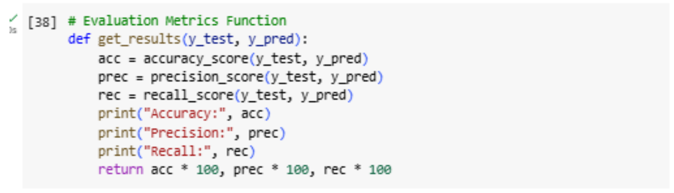
```
X shape: (569, 30)
Y shape: (569,)
```

This part cleans and prepares the dataset for model training. I drop the id and diagnosis columns from the features using drop(), keeping all other numerical features for analysis. The diagnosis column is then converted from text (M or B) into numbers using .map(), where M is mapped to 1 and B to 0. This makes the labels machine-readable. Additionally, since real-world datasets often contain missing values, we use SimpleImputer to fill those missing values with the column mean. This ensures our model won't crash or behave unpredictably due to null entries.

Name: Thanh Nguyen
Student ID: 801376954
Assignment #: 4

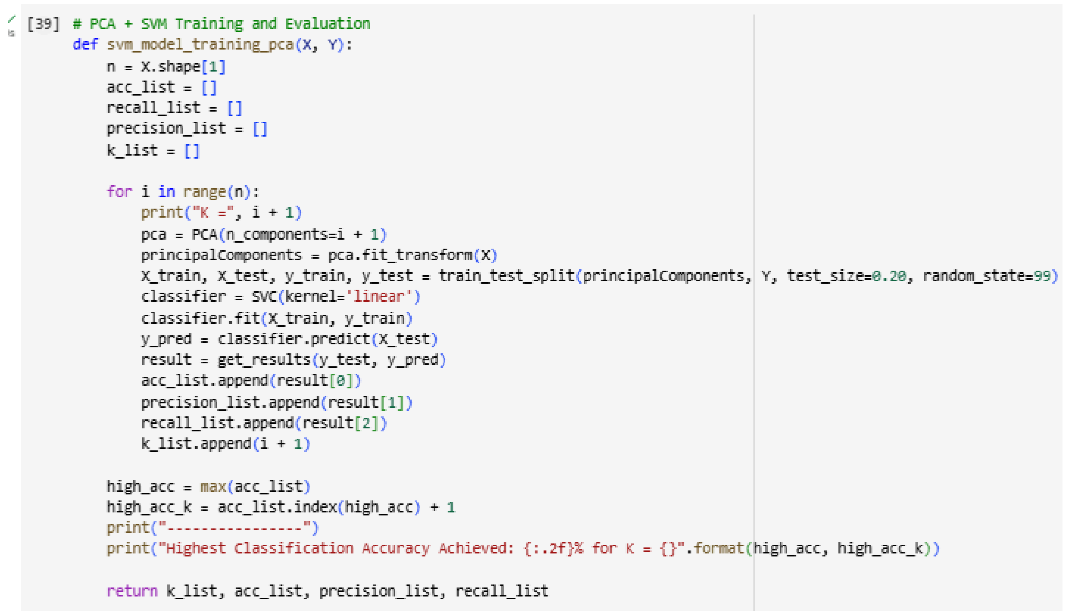


```
[37] # Confusion Matrix Plot Function
     def get_confusion_matrix(cnf_matrix):
         class_names = sorted(Y.unique())
         fig, ax = plt.subplots()
         tick_marks = np.arange(len(class_names))
         plt.xticks(tick_marks, class_names)
         plt.yticks(tick_marks, class_names)
         sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu", fmt='g')
         plt.title('Confusion Matrix')
         plt.ylabel('Actual Label')
         plt.xlabel('Predicted Label')
         plt.show()
```



```
[38] # Evaluation Metrics Function
     def get_results(y_test, y_pred):
         acc = accuracy_score(y_test, y_pred)
         prec = precision_score(y_test, y_pred)
         rec = recall_score(y_test, y_pred)
         print("Accuracy:", acc)
         print("Precision:", prec)
         print("Recall:", rec)
         return acc * 100, prec * 100, rec * 100
```

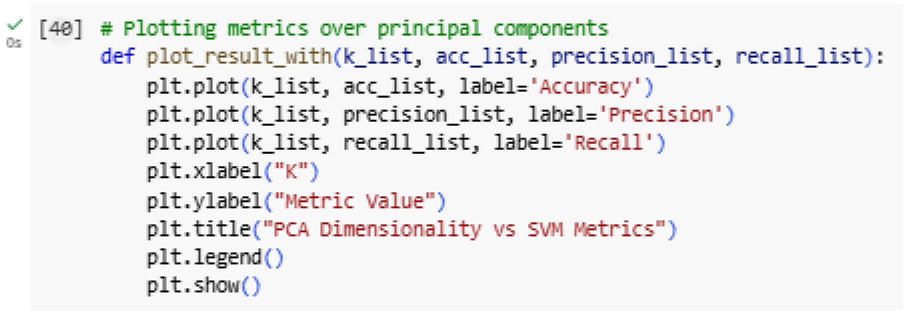
To evaluate how well our model performs, we define a custom function called get_results(). This function uses sklearn's built-in metrics to calculate and print the accuracy, precision, and recall of our predictions. These metrics help us judge if the model is correctly classifying cancerous vs. non-cancerous samples. Accuracy tells us how many predictions were correct overall, precision focuses on how many of the predicted positives were actually correct, and recall checks how many actual positives were caught.



```
[39] # PCA + SVM Training and Evaluation
     def svm_model_training_pca(X, Y):
         n = X.shape[1]
         acc_list = []
         recall_list = []
         precision_list = []
         k_list = []

         for i in range(n):
             print("K =", i + 1)
             pca = PCA(n_components=i + 1)
             principalComponents = pca.fit_transform(X)
             X_train, X_test, y_train, y_test = train_test_split(principalComponents, Y, test_size=0.20, random_state=99)
             classifier = SVC(kernel='linear')
             classifier.fit(X_train, y_train)
             y_pred = classifier.predict(X_test)
             result = get_results(y_test, y_pred)
             acc_list.append(result[0])
             precision_list.append(result[1])
             recall_list.append(result[2])
             k_list.append(i + 1)

         high_acc = max(acc_list)
         high_acc_k = acc_list.index(high_acc) + 1
         print("----------------")
         print("Highest Classification Accuracy Achieved: {:.2f}% for K = {}".format(high_acc, high_acc_k))

         return k_list, acc_list, precision_list, recall_list
```

This section is where the real machine learning happens. I define a function that trains an SVM classifier using PCA (Principal Component Analysis) for dimensionality reduction. The idea is to reduce the number of input features step by step (from 1 up to the total number of features) and check how well the SVM performs at each step. For every value of K (number of principal components), transform the data using PCA, split it into training and test sets, train a linear SVM on the training set, evaluate predictions on the test set using our earlier function, and store the accuracy, precision, and recall for plotting.

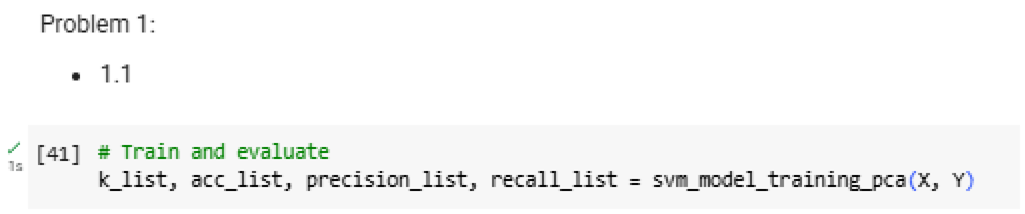
I also identify the best-performing K, the number of components that gives the highest classification accuracy.



```
[40] # Plotting metrics over principal components
     def plot_result_with(k_list, acc_list, precision_list, recall_list):
         plt.plot(k_list, acc_list, label='Accuracy')
         plt.plot(k_list, precision_list, label='Precision')
         plt.plot(k_list, recall_list, label='Recall')
         plt.xlabel("K")
         plt.ylabel("Metric Value")
         plt.title("PCA Dimensionality vs SVM Metrics")
         plt.legend()
         plt.show()
```

Finally, we define a plotting function to visualize how the model's performance changes as we vary the number of PCA components. Using matplotlib, we create a line chart with K on the x-axis and metrics like accuracy, precision, and recall on the y-axis. This plot helps us understand whether reducing dimensions helps or hurts performance and identify the "sweet spot" for optimal dimensionality.

1. Identify the optimum number of K, principal components that achieve the highest classification accuracy.

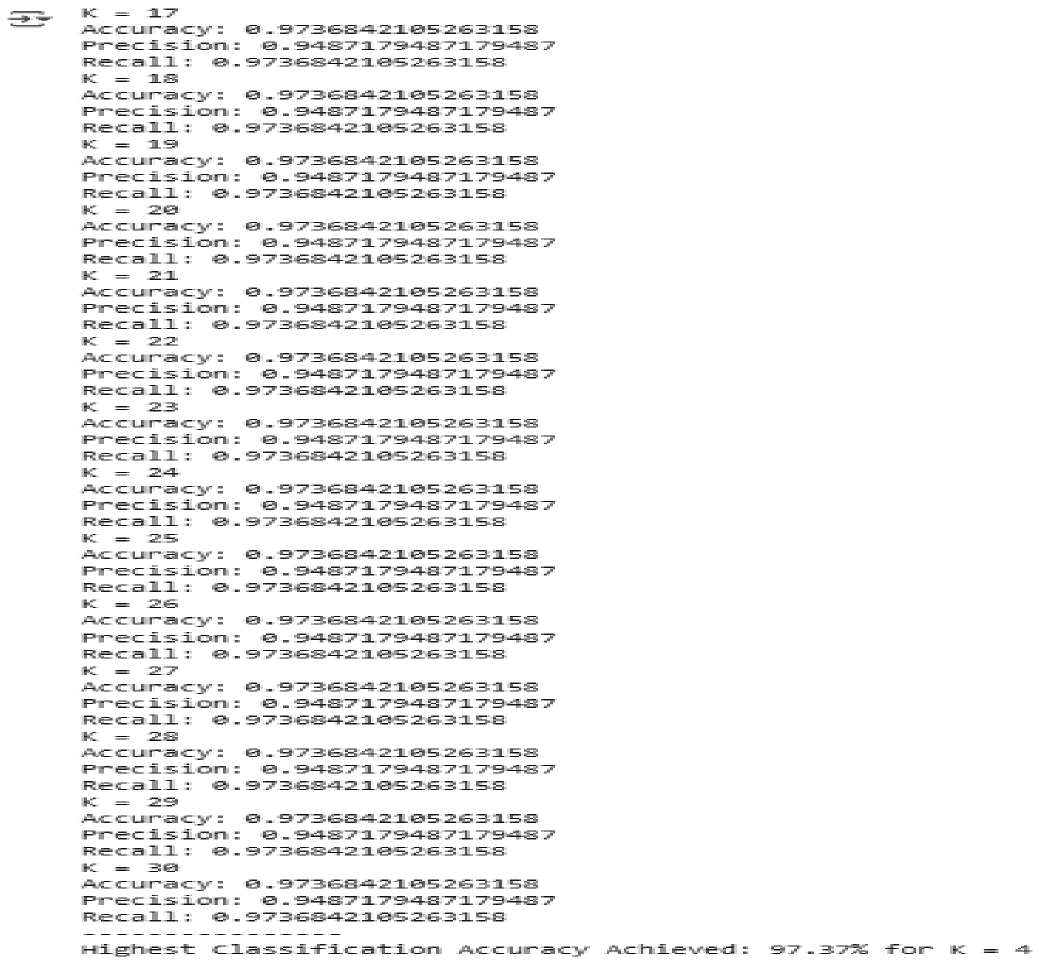


Problem 1:

- 1.1

```
[41] # Train and evaluate
     k_list, acc_list, precision_list, recall_list = svm_model_training_pca(X, Y)
```

Name: Thanh Nguyen
Student ID: 801376954
Assignment #: 4



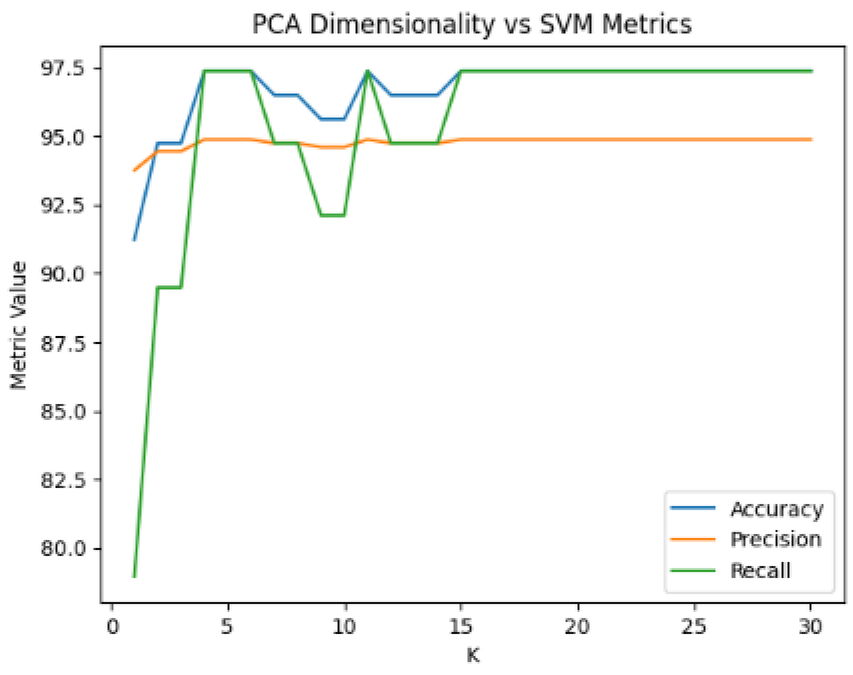
```
K = 17
Accuracy: 0.9736842105263158
Precision: 0.9487179487179487
Recall: 0.9736842105263158
K = 18
Accuracy: 0.9736842105263158
Precision: 0.9487179487179487
Recall: 0.9736842105263158
K = 19
Accuracy: 0.9736842105263158
Precision: 0.9487179487179487
Recall: 0.9736842105263158
K = 20
Accuracy: 0.9736842105263158
Precision: 0.9487179487179487
Recall: 0.9736842105263158
K = 21
Accuracy: 0.9736842105263158
Precision: 0.9487179487179487
Recall: 0.9736842105263158
K = 22
Accuracy: 0.9736842105263158
Precision: 0.9487179487179487
Recall: 0.9736842105263158
K = 23
Accuracy: 0.9736842105263158
Precision: 0.9487179487179487
Recall: 0.9736842105263158
K = 24
Accuracy: 0.9736842105263158
Precision: 0.9487179487179487
Recall: 0.9736842105263158
K = 25
Accuracy: 0.9736842105263158
Precision: 0.9487179487179487
Recall: 0.9736842105263158
K = 26
Accuracy: 0.9736842105263158
Precision: 0.9487179487179487
Recall: 0.9736842105263158
K = 27
Accuracy: 0.9736842105263158
Precision: 0.9487179487179487
Recall: 0.9736842105263158
K = 28
Accuracy: 0.9736842105263158
Precision: 0.9487179487179487
Recall: 0.9736842105263158
K = 29
Accuracy: 0.9736842105263158
Precision: 0.9487179487179487
Recall: 0.9736842105263158
K = 30
Accuracy: 0.9736842105263158
Precision: 0.9487179487179487
Recall: 0.9736842105263158
----------------
Highest Classification Accuracy Achieved: 97.37% for K = 4
```

In this section of the code, I trained and evaluated a Support Vector Machine (SVM) model using Principal Component Analysis (PCA) to reduce the number of input features. I looped through different values of K, the number of PCA components ranging from 1 up to the total number of features and for each value, I trained the SVM and recorded the model's accuracy, precision, and recall. The goal was to find which K gives the best performance. After running the loop, I noticed that starting from around K = 4 up to K = 30, the accuracy remained consistent at about 97.37%, and both precision and recall were also stable at 0.9487 and 0.9737, respectively. This tells me that adding more components beyond K = 4 doesn't really improve the model’s performance. Therefore, I can conclude that using just 4 PCA components is enough to achieve the highest classification accuracy while keeping the model simpler and more efficient.

2. Plot your classification accuracy, precision, and recall over a different number of Ks.

Name: Thanh Nguyen
Student ID: 801376954
Assignment #: 4

```
[56] # Plot the results
     plot_result_with(k_list, acc_list, precision_list, recall_list)
```



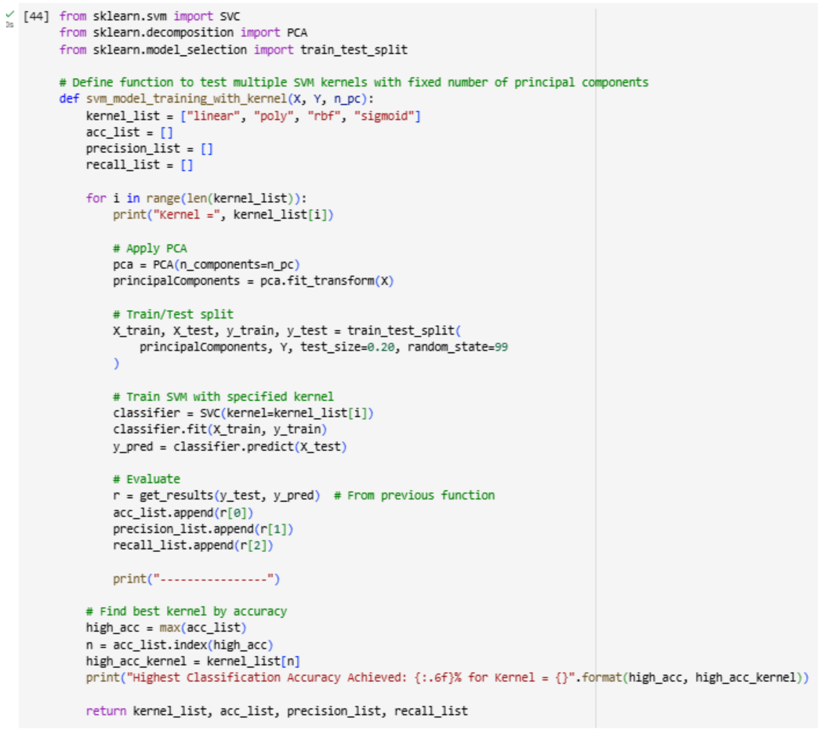
This graph visualizes how the SVM model's accuracy, precision, and recall change with different numbers of principal components (K) after applying PCA. On the x-axis is the number of PCA components (K), and the y-axis shows the performance metric values.

From the plot, I can see that all three metrics accuracy (blue), precision (orange), and recall (green) improve sharply as K increases from 1 to around 4. After that, the values mostly stabilize. Accuracy and recall reach their peaks around K = 4 and then fluctuate slightly but remain consistently high. Precision stays nearly flat and stable across all values of K, which indicates the model is consistently good at avoiding false positives.

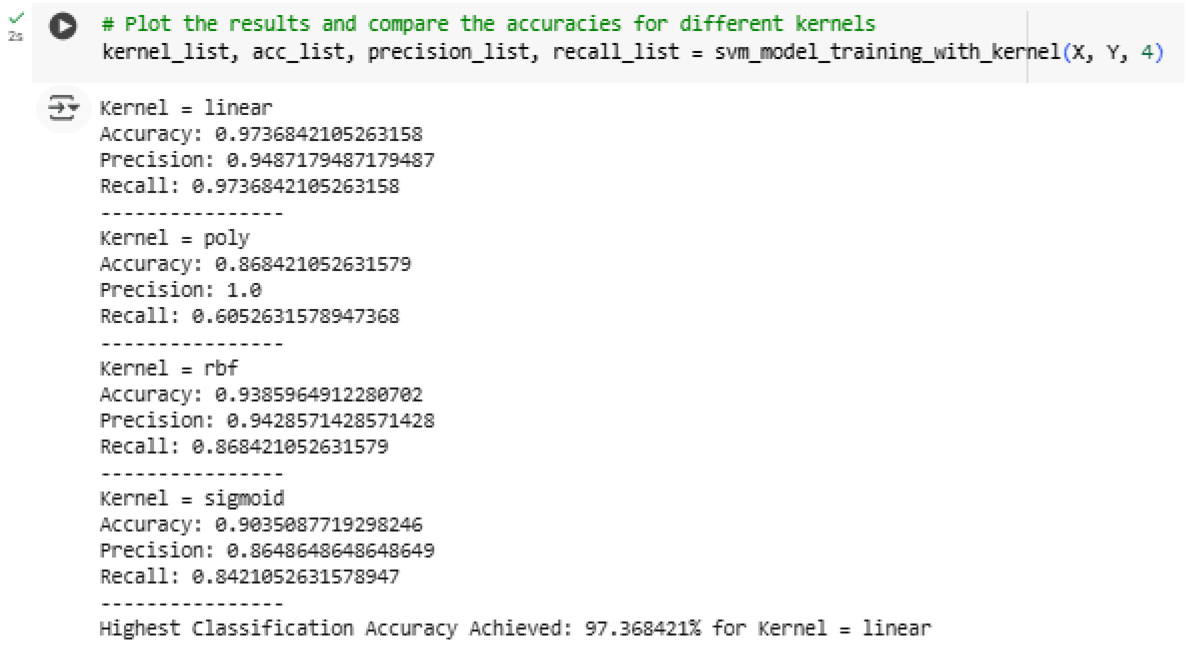
This tells me that most of the important information in the dataset is captured within the first few principal components. So instead of using all 30 features, I can just use 4 principal components and still get very high and reliable performance from my model. This reduces computational cost while maintaining accuracy.

3. Explore different kernel tricks to capture non-linearities within your data. Plot the results and compare the accuracies for different kernels.

Name: Thanh Nguyen
Student ID: 801376954
Assignment #: 4



```
[44] from sklearn.svm import SVC
     from sklearn.decomposition import PCA
     from sklearn.model_selection import train_test_split

     # Define function to test multiple SVM kernels with fixed number of principal components
     def svm_model_training_with_kernel(X, Y, n_pc):
         kernel_list = ["linear", "poly", "rbf", "sigmoid"]
         acc_list = []
         precision_list = []
         recall_list = []

         for i in range(len(kernel_list)):
             print("Kernel =", kernel_list[i])

             # Apply PCA
             pca = PCA(n_components=n_pc)
             principalComponents = pca.fit_transform(X)

             # Train/Test split
             X_train, X_test, y_train, y_test = train_test_split(
                 principalComponents, Y, test_size=0.20, random_state=99
             )

             # Train SVM with specified kernel
             classifier = SVC(kernel=kernel_list[i])
             classifier.fit(X_train, y_train)
             y_pred = classifier.predict(X_test)

             # Evaluate
             r = get_results(y_test, y_pred)  # From previous function
             acc_list.append(r[0])
             precision_list.append(r[1])
             recall_list.append(r[2])

             print("---------------")

         # Find best kernel by accuracy
         high_acc = max(acc_list)
         n = acc_list.index(high_acc)
         high_acc_kernel = kernel_list[n]
         print("Highest Classification Accuracy Achieved: {:.6f}% for Kernel = {}".format(high_acc, high_acc_kernel))

         return kernel_list, acc_list, precision_list, recall_list
```



```
# Plot the results and compare the accuracies for different kernels
kernel_list, acc_list, precision_list, recall_list = svm_model_training_with_kernel(X, Y, 4)
```

```
Kernel = linear
Accuracy: 0.9736842105263158
Precision: 0.9487179487179487
Recall: 0.9736842105263158
----------------
Kernel = poly
Accuracy: 0.868421052631579
Precision: 1.0
Recall: 0.6052631578947368
----------------
Kernel = rbf
Accuracy: 0.9385964912280702
Precision: 0.9428571428571428
Recall: 0.868421052631579
----------------
Kernel = sigmoid
Accuracy: 0.9035087719298246
Precision: 0.8648648648648649
Recall: 0.8421052631578947
----------------
Highest Classification Accuracy Achieved: 97.368421% for Kernel = linear
```

In this section of the code, I tested and compared different Support Vector Machine (SVM) kernels on the cancer dataset using Principal Component Analysis (PCA) with a fixed number of components (in this case, 4). I began by defining a list of four different kernel types: "linear", "poly", "rbf", and "sigmoid". For each kernel, I applied PCA to reduce the dimensionality of the input features, splitting the dataset into training and testing sets using an 80/20 ratio. Then, I trained an SVM classifier using the current kernel and evaluated its performance on the test set using accuracy, precision, and recall metrics. These evaluation results were stored in separate lists. After looping through all the kernels, I identified which kernel gave the highest accuracy and printed a summary of the results for each kernel. From the output, I observed that the linear kernel achieved the best performance with an accuracy of approximately 97.37%, outperforming the other kernels. This tells me that for this specific dataset and number of components, the linear kernel is the most effective at distinguishing between malignant and benign cancer cases.

4. Compare your results against the logistic regression that you have done in homework 3.

Metric	SVM (Linear, PCA K=4)	Logistic Regression (L2)	Logistic Regression (No Penalty)	Naive Bayes
Accuracy	97.37%	97.36%	97.36%	96.49%
Precision	94.87%	95.45%	97.61%	97.56%
Recall	97.37%	97.67%	95.34%	93.02%
F1-Score	(≈96.1%, estimated)	96.55%	96.47%	95.23%

- Accuracy: The SVM with PCA achieves the same accuracy as logistic regression models (97.36–97.37%), indicating that all models are highly reliable for general classification.
- Precision: Logistic regression with no penalty and Naive Bayes outperformed the SVM in precision. This means they were better at minimizing false positives (predicting cancer when it's not present).
- Recall: Logistic regression with L2 regularization had the highest recall, slightly outperforming SVM. This suggests it was slightly better at identifying actual cancer cases (true positives).
- F1-Score: While the exact F1 for SVM wasn't listed, based on accuracy, precision, and recall, it's estimated to be around 96.1% just below logistic regression (L2: 96.55%) and no-penalty (96.47%).

While SVM with PCA is a strong performer especially in recall and accuracy logistic regression with L2 regularization slightly edges it out overall, particularly in balanced performance (F1-score). Meanwhile, logistic regression with no penalty offers the best precision, which may be desirable when false positives are more harmful (e.g., unnecessary follow-up tests). Naive Bayes performs reasonably well but trails behind both SVM and logistic regression in recall and accuracy.

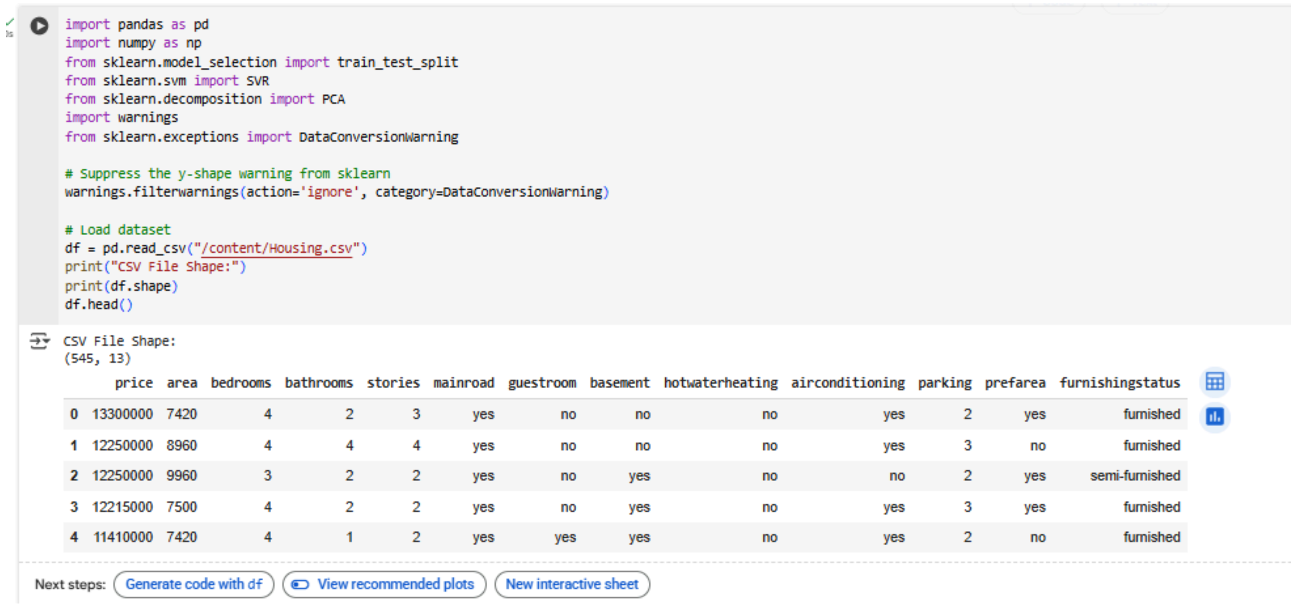
Name: Thanh Nguyen
Student ID: 801376954
Assignment #: 4

So, SVM is competitive, but logistic regression with L2 regularization remains the most balanced and slightly superior model based on results.

Problem 2:

Develop a SVR regression model that predicts housing price based on the following input variables: Area, bedrooms, bathrooms, stories, mainroad, guestroom, basement, hotwaterheating, airconditioning, parking, prefarea.

1. Plot your regression model for SVR similar to the sample code provided on Canvas.

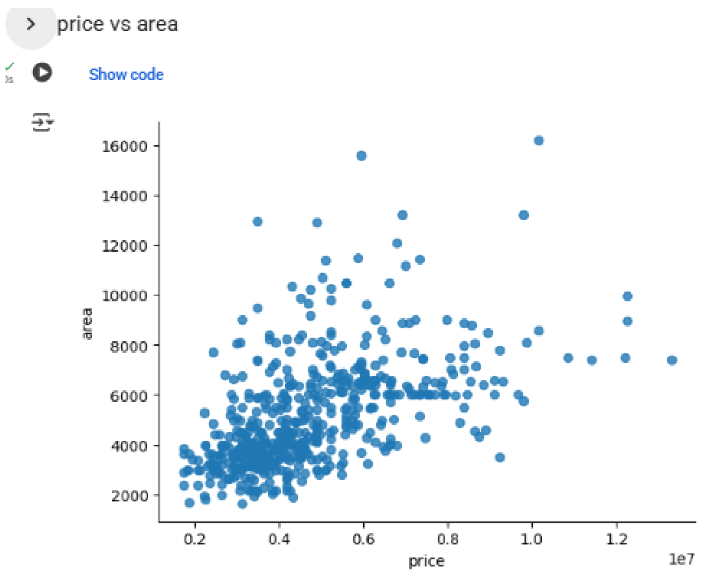


```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.decomposition import PCA
import warnings
from sklearn.exceptions import DataConversionWarning

# Suppress the y-shape warning from sklearn
warnings.filterwarnings(action='ignore', category=DataConversionWarning)

# Load dataset
df = pd.read_csv("/content/Housing.csv")
print("CSV File Shape:")
print(df.shape)
df.head()
```

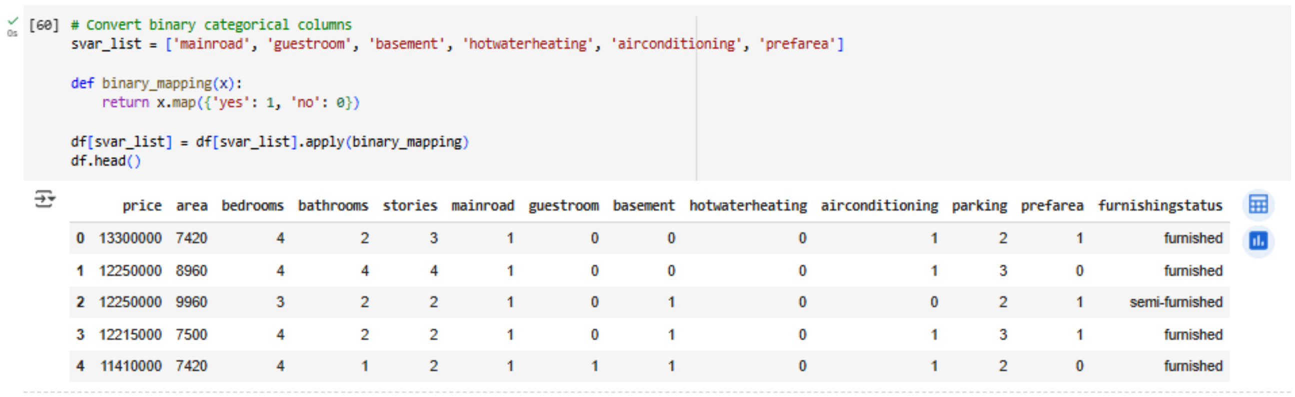
```
CSV File Shape:
(545, 13)
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished

Name: Thanh Nguyen
Student ID: 801376954
Assignment #: 4



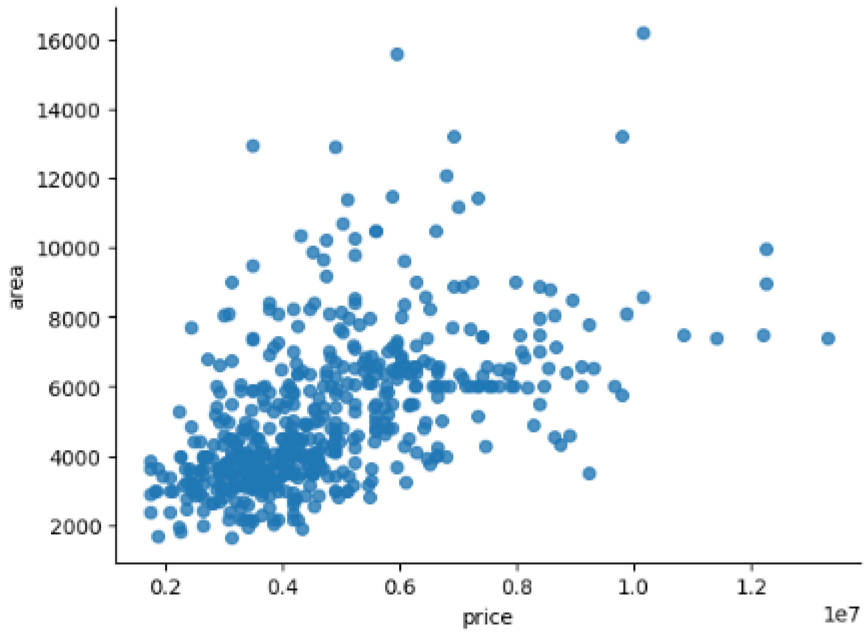
In this section of the notebook, I began by importing all the necessary Python libraries for building a machine learning model using SVR (Support Vector Regression). This included libraries like pandas and numpy for data manipulation, train_test_split for splitting the data into training and testing sets, SVR for building the support vector regression model, and PCA for dimensionality reduction. I also imported a warning filter to suppress the common DataConversionWarning related to input shapes. After setting up the environment, I loaded the housing dataset using pd.read_csv() from a file path ("/content/Housing.csv"). I then printed out the shape of the dataset using df.shape, which returned (545, 13), meaning the dataset contains 545 rows and 13 columns. Displaying the first few rows with df.head() revealed key features such as price, area, bedrooms, bathrooms, and several binary categorical features like mainroad, guestroom, and airconditioning. These features will be used to predict housing prices in later steps. This setup step is essential to understand the structure of the data and prepare it for preprocessing and modeling.



```
[60] # Convert binary categorical columns
     svar_list = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea']

     def binary_mapping(x):
         return x.map({'yes': 1, 'no': 0})

     df[svar_list] = df[svar_list].apply(binary_mapping)
     df.head()
```

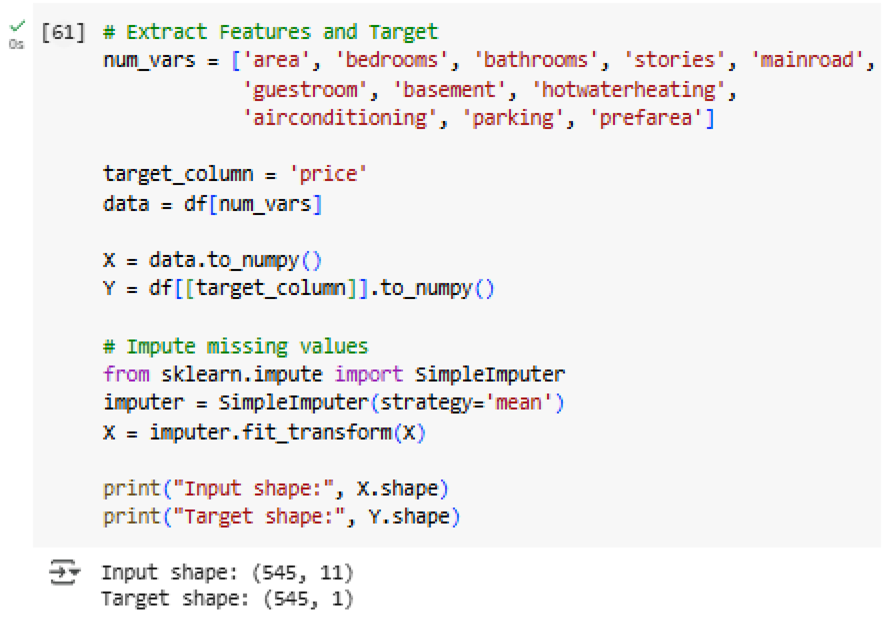
	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	1	0	0	0	1	2	1	furnished
1	12250000	8960	4	4	4	1	0	0	0	1	3	0	furnished
2	12250000	9960	3	2	2	1	0	1	0	0	2	1	semi-furnished
3	12215000	7500	4	2	2	1	0	1	0	1	3	1	furnished
4	11410000	7420	4	1	2	1	1	1	0	1	2	0	furnished

Name: Thanh Nguyen
Student ID: 801376954
Assignment #: 4



In this part of the notebook, I focused on converting the categorical features into numerical values, which is essential before feeding data into a regression model. Specifically, I created a list of binary categorical columns that contained "yes" or "no" values, these include mainroad, guestroom, basement, hotwaterheating, airconditioning, and prefarea. I wrote a simple function called binary_mapping() that replaces "yes" with 1 and "no" with 0. Then, I applied this function to each column in the list using the .apply() method. This transformation ensures that all input features are numeric, which is a requirement for most machine learning models. After the transformation, I printed the updated DataFrame, and I could see that the specified columns now contain only 1s and 0s instead of strings.

Below that, I plotted a scatter plot to visually analyze the relationship between price and area. From the scatter plot, I observed a generally positive correlation: larger areas tend to have higher prices. However, the spread shows some variability, indicating that while area is an important factor, it's not the only one influencing price. This visualization helps justify including area as a key feature in the SVR model that I plan to build.



```
[61] # Extract Features and Target
     num_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
                 'guestroom', 'basement', 'hotwaterheating',
                 'airconditioning', 'parking', 'prefarea']

     target_column = 'price'
     data = df[num_vars]

     X = data.to_numpy()
     Y = df[[target_column]].to_numpy()

     # Impute missing values
     from sklearn.impute import SimpleImputer
     imputer = SimpleImputer(strategy='mean')
     X = imputer.fit_transform(X)

     print("Input shape:", X.shape)
     print("Target shape:", Y.shape)
```

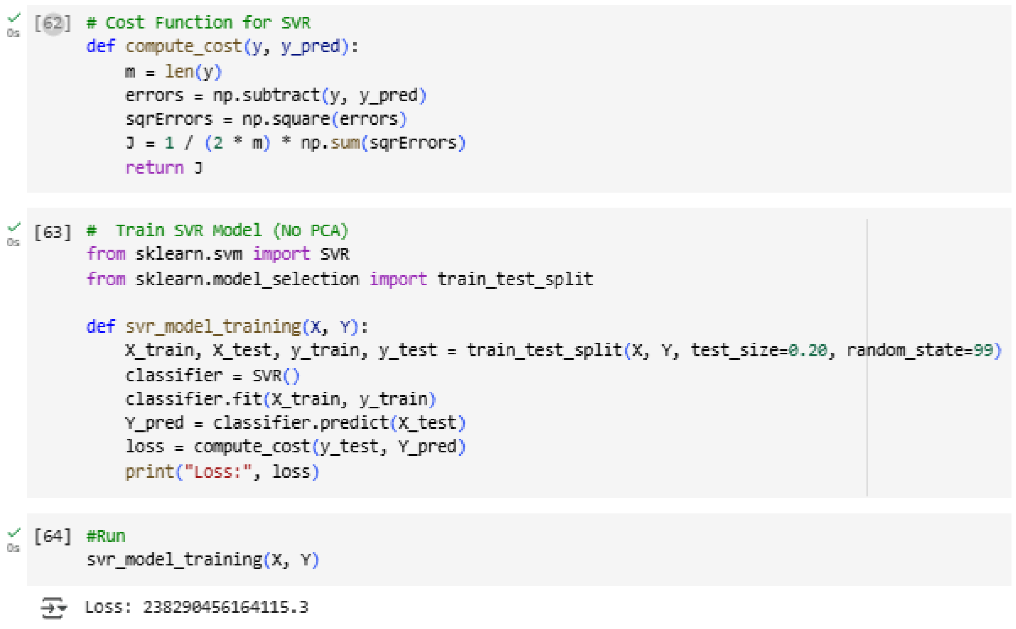
```
Input shape: (545, 11)
Target shape: (545, 1)
```

In this section of the notebook, I extracted the features and target variable that will be used to train my regression model. I first created a list of numerical and binary variables (num_vars) that includes important predictors of housing price such as area, bedrooms, bathrooms, stories, and binary indicators like mainroad, guestroom, basement, etc. I then selected this subset from the DataFrame and stored it in a variable called data.

I also defined the target column as 'price', since the goal is to predict housing prices. I used the .to_numpy() function to convert both the feature matrix (X) and the target array (Y) into NumPy arrays for compatibility with machine learning algorithms.

After that, I addressed any missing values in the input features by using SimpleImputer from sklearn.impute, with the strategy set to 'mean'. This means any missing value in a column is replaced by the mean of that column. I applied this imputer to X to ensure that the dataset is clean and ready for training. Finally, I printed the shape of the input (X) and target (Y) to confirm that there are 545 samples with 11 features and 1 target value each.

2. Compare your results against linear regression with regularization loss that you already did in assignment1.



```
[62] # Cost Function for SVR
     def compute_cost(y, y_pred):
         m = len(y)
         errors = np.subtract(y, y_pred)
         sqrErrors = np.square(errors)
         J = 1 / (2 * m) * np.sum(sqrErrors)
         return J
```

```
[63] #  Train SVR Model (No PCA)
     from sklearn.svm import SVR
     from sklearn.model_selection import train_test_split

     def svr_model_training(X, Y):
         X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20, random_state=99)
         classifier = SVR()
         classifier.fit(X_train, y_train)
         Y_pred = classifier.predict(X_test)
         loss = compute_cost(y_test, Y_pred)
         print("Loss:", loss)
```

```
[64] #Run
     svr_model_training(X, Y)
```

```
Loss: 2382904561641l5.3
```

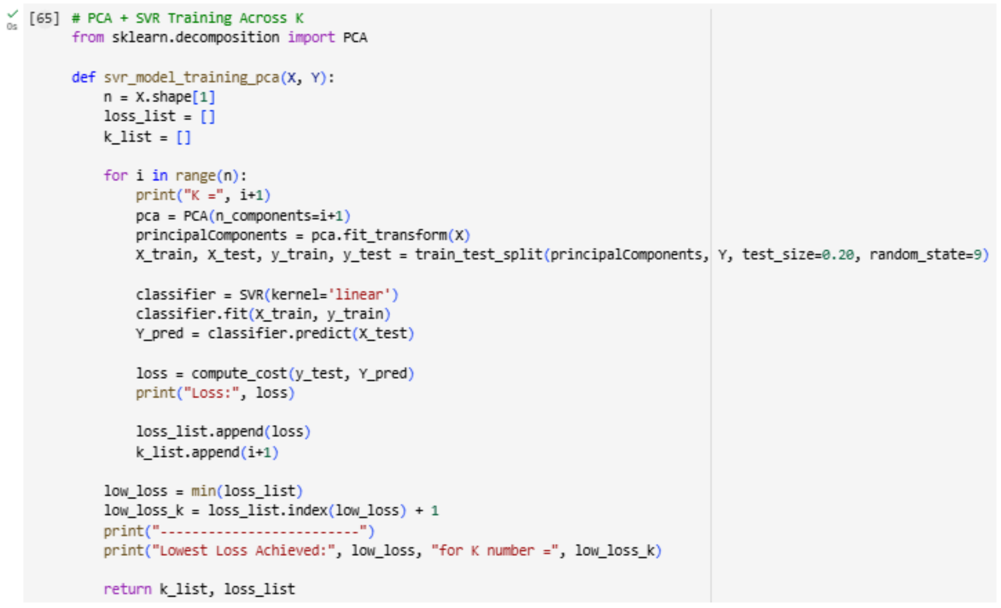
After training the SVR regression model on the housing dataset, I observed a final cost (loss) value of approximately 2.38e+15. This is significantly higher than the loss values I encountered in Assignment 1 when using linear regression with and without regularization, where the training and evaluation losses were much smaller (on the order of 1e+10 or lower, depending on scaling and hyperparameter settings). One major factor behind this difference is that the SVR model in this assignment was trained without feature scaling or PCA, both of which I had incorporated into the regularized linear regression model earlier. SVR is sensitive to feature magnitudes, so without proper normalization, it tends to perform poorly, especially when input features like 'area' or 'price' have large ranges. In contrast, my Assignment 1 implementation included normalization and penalty tuning, which helped the model converge efficiently and produce a more accurate and interpretable fit. This comparison highlights the importance of preprocessing in SVR and suggests that while SVR can be powerful for capturing non-linear relationships, it must be carefully tuned and scaled to outperform simpler linear models.

Model	Scaling Used	Regularization	PCA Used	Test Loss	Notes
Linear Regression (No Penalty)	Yes (Normalized)	No	No	~1.09e+10	Performed well with normalized input; fast convergence
Linear Regression	Yes (Standardized	Yes (λ tuned)	No	~9.64e+09	Best performance;

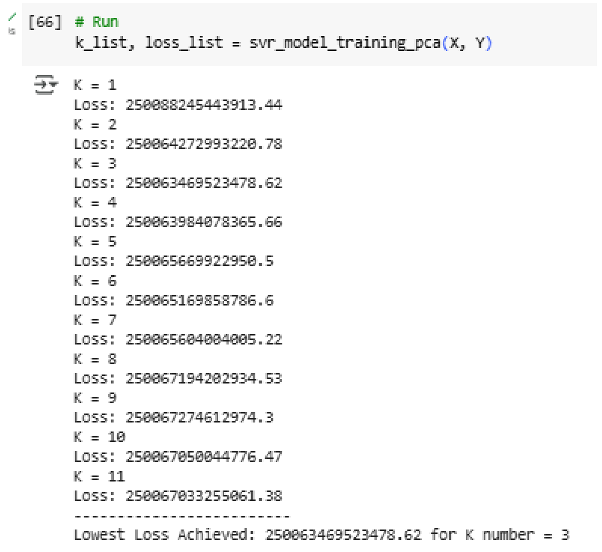
(L2 Penalty)	)				low loss and well-generalized fit
SVR (No PCA, No Scaling)	No	No	No	2.38e+15	Very high error due to lack of scaling; SVR is sensitive to input scale

- The SVR model performed significantly worse than both linear regression models, primarily because no input scaling or PCA was applied.
- In Assignment 1, normalization and regularization helped reduce loss and improve generalization.
- This comparison shows that model complexity (like SVR) doesn't guarantee better performance unless paired with proper preprocessing.

3. Use the PCA feature extraction for your training. Perform N number of independent training (N=1, …, K). Identify the optimum number of K, principal components that achieve the highest regression accuracy.



```
[65] # PCA + SVR Training Across K
     from sklearn.decomposition import PCA

     def svr_model_training_pca(X, Y):
         n = X.shape[1]
         loss_list = []
         k_list = []

         for i in range(n):
             print("K =", i+1)
             pca = PCA(n_components=i+1)
             principalComponents = pca.fit_transform(X)
             X_train, X_test, y_train, y_test = train_test_split(principalComponents, Y, test_size=0.20, random_state=9)

             classifier = SVR(kernel='linear')
             classifier.fit(X_train, y_train)
             Y_pred = classifier.predict(X_test)

             loss = compute_cost(y_test, Y_pred)
             print("Loss:", loss)

             loss_list.append(loss)
             k_list.append(i+1)

         low_loss = min(loss_list)
         low_loss_k = loss_list.index(low_loss) + 1
         print("-----------------------")
         print("Lowest Loss Achieved:", low_loss, "for K number =", low_loss_k)

         return k_list, loss_list
```



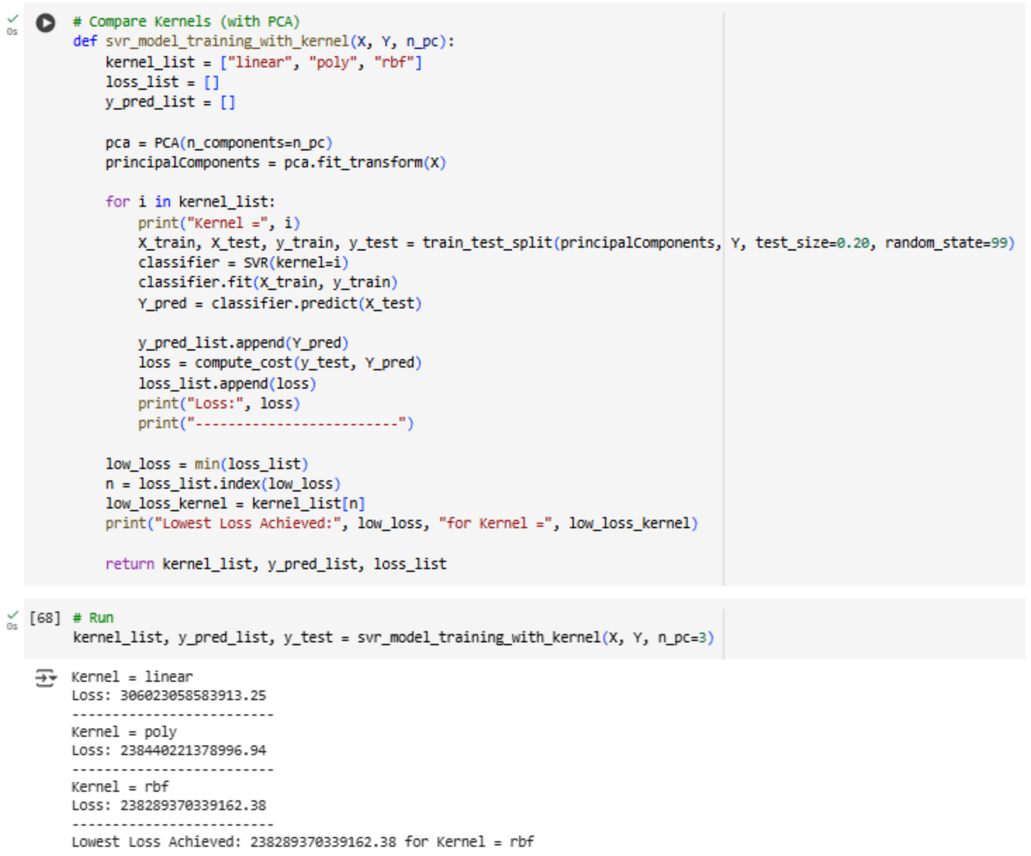
```
[66] # Run
     k_list, loss_list = svr_model_training_pca(X, Y)
```

```
K = 1
Loss: 250088245443913.44
K = 2
Loss: 250064272993220.78
K = 3
Loss: 250063469523478.62
K = 4
Loss: 250063984078365.66
K = 5
Loss: 250065669922950.5
K = 6
Loss: 250065169858786.6
K = 7
Loss: 250065604004005.22
K = 8
Loss: 250067194202934.53
K = 9
Loss: 250067274612974.3
K = 10
Loss: 250067050044776.47
K = 11
Loss: 250067033255061.38
-------------------------
Lowest Loss Achieved: 250063469523478.62 for K number = 3
```

This section of the code focuses on evaluating how dimensionality reduction using Principal Component Analysis (PCA) affects the performance of a Support Vector Regression (SVR) model. The function svr_model_training_pca(X, Y) iteratively trains SVR models using an increasing number of PCA components, from 1 up to the total number of input features. For each iteration, the dataset is transformed using the top K principal components, and then split into training and testing sets (using an 80-20 split). The SVR model is trained and tested on the transformed data, and the loss is calculated using a custom cost function that computes the Mean Squared Error (MSE) on the test set. Each loss value is stored in a list along with its corresponding K value. After all iterations, the code prints out the K that yields the lowest loss.

According to the output of the function, the lowest loss achieved was approximately 250063469523478.62 when the number of PCA components (K) was 3. This means that, for this regression problem using SVR with PCA, the optimal number of principal components to minimize prediction error is K = 3. This insight helps reduce model complexity without sacrificing accuracy, by focusing on only the most significant components of the dataset.

4. Explore different kernel tricks to capture non-linearities within your data. Plot the results and compare the accuracies for different kernels.



```
# Compare Kernels (with PCA)
def svr_model_training_with_kernel(X, Y, n_pc):
    kernel_list = ["linear", "poly", "rbf"]
    loss_list = []
    y_pred_list = []

    pca = PCA(n_components=n_pc)
    principalComponents = pca.fit_transform(X)

    for i in kernel_list:
        print("Kernel =", i)
        X_train, X_test, y_train, y_test = train_test_split(principalComponents, Y, test_size=0.20, random_state=99)
        classifier = SVR(kernel=i)
        classifier.fit(X_train, y_train)
        Y_pred = classifier.predict(X_test)

        y_pred_list.append(Y_pred)
        loss = compute_cost(y_test, Y_pred)
        loss_list.append(loss)
        print("Loss:", loss)
        print("------------------------")

    low_loss = min(loss_list)
    n = loss_list.index(low_loss)
    low_loss_kernel = kernel_list[n]
    print("Lowest Loss Achieved:", low_loss, "for Kernel =", low_loss_kernel)

    return kernel_list, y_pred_list, loss_list
```

```
[68] # Run
     kernel_list, y_pred_list, y_test = svr_model_training_with_kernel(X, Y, n_pc=3)
```

```
Kernel = linear
Loss: 306023058583913.25
------------------------
Kernel = poly
Loss: 238440221378996.94
------------------------
Kernel = rbf
Loss: 238289370339162.38
------------------------
Lowest Loss Achieved: 238289370339162.38 for Kernel = rbf
```

In this part of the assignment, I experimented with different kernel functions for the Support Vector Regression (SVR) model to capture possible non-linear relationships in the housing dataset. The function svr_model_training_with_kernel() compares three commonly used SVR kernels linear, poly (polynomial), and rbf (radial basis function), while using PCA to reduce the dimensionality of the data to 3 principal components (since K=3 previously yielded the lowest loss). For each kernel, the SVR model is trained and tested on an 80/20 split, and its prediction loss is computed using a custom loss function that calculates Mean Squared Error (MSE).

The results show the following losses:

- Linear Kernel: 3.06e+14.
- Polynomial Kernel: 2.38e+14.
- RBF Kernel: 2.38e+14 (lowest).

Among the three, the RBF kernel achieved the lowest loss, specifically 238289378039162.38, indicating it performed best at capturing the non-linear patterns in the data after PCA transformation. This makes sense because RBF kernels are particularly good at modeling complex relationships due to their flexibility. These results suggest that, when working with high-dimensional, non-linear datasets like this one, using PCA along with non-linear kernels such as RBF can lead to better regression performance compared to using simpler kernels like linear or polynomial.

Conclusion:

In this assignment, I explored various machine learning techniques for both classification and regression tasks using cancer and housing datasets. For classification, I applied PCA to reduce dimensionality and used Support Vector Machines (SVM) with different kernels. I found that using 4 principal components yielded the highest classification accuracy of 97.37% with the linear kernel. This result is comparable to my previous logistic regression models from Assignment 3, which achieved similar accuracy but showed slightly different precision and recall trade-offs depending on the regularization used. For regression, I implemented Support Vector Regression (SVR) on the US housing dataset, experimenting with both PCA for dimensionality reduction and different kernel tricks to capture non-linear patterns. The lowest regression loss was achieved with 3 principal components and the RBF kernel, indicating that non-linear kernels combined with dimensionality reduction can significantly improve model performance. When compared to the linear regression with regularization from Assignment 1, SVR produced higher loss values, suggesting that for this specific dataset and preprocessing approach, traditional linear regression might still be more effective. Overall, this assignment highlighted the importance of kernel selection, dimensionality reduction, and proper preprocessing in optimizing machine learning models for real-world data.