 with st.spinner('Please wait while we conduct principal component analysis'):

                    my\_bar = st.progress(0)

                    time.sleep(10)

                    for percent\_complete in range(100):

                        time.sleep(0.01)

                        my\_bar.progress(percent\_complete + 1)

                    start\_time = time.time()

                    ### From Jupyter - Principal component analysis

                    # Initially, visualize the important data features

                    # Scale the features

                    # Separating out the features

                    x = df.iloc[:, 1:-1].sample(10000).values #subsampling for efficiency and speed

                    # Separating out the target

                    y = df.iloc[:,0].sample(10000).values #subsampling for efficiency and speed

                    # Standardizing the features

                    x = StandardScaler().fit\_transform(x)

                    # Dimensionality reduction

                    from sklearn.decomposition import PCA

                    pca = PCA(n\_components=10)

                    principalComponents = pca.fit\_transform(x)

                    principalDf = pd.DataFrame(data = principalComponents

                                , columns = ['principal component 1', 'principal component 2', 'principal component 3', 'principal component 4', 'principal component 5', 'principal component 6', 'principal component 7', 'principal component 8', 'principal component 9', 'principal component 10'])

                    # Concatenate DF across axis 1

                    finalDf = pd.concat([principalDf, df['BRAND']], axis = 1)

                    st.write("Table of top 10 principal components")

                    st.write(finalDf)

                    # Plot 2D data

                    fig = plt.figure(figsize = (8,8))

                    ax = fig.add\_subplot(1,1,1)

                    ax.set\_xlabel('Principal Component 1', fontsize = 15)

                    ax.set\_ylabel('Principal Component 2', fontsize = 15)

                    ax.set\_title('PCA showing top 2 components', fontsize = 20)

                    targets = ['BRAND']

                    colors = ['r', 'g', 'b']

                    for target, color in zip(targets,colors):

                        indicesToKeep = finalDf['BRAND'] == target

                        ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']

                                , finalDf.loc[indicesToKeep, 'principal component 2']

                                , c = color

                                , s = 50)

                        # ax.set\_xticks([0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1])

                        # ax.set\_yticks([0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1])

                        ax.set\_xticks([-100, -10, -0.1, 0, 0.1, 1, 10, 100])

                        ax.set\_yticks([-100, -10, -0.1, 0, 0.1, 1, 10, 100])

                    ax.legend(targets)

                    ax.grid()

                    buf = BytesIO()

                    fig.savefig(buf, format="png")

                    st.image(buf)

                    # Explain the variance

                    st.write("Explained variance from top 10 components:")

                    st.write(pca.explained\_variance\_ratio\_)

                    ### End

                    st.text("") # Spacer

                    st.write("")

                    st.write("Principal component analysis took ", time.time() - start\_time, "seconds to run")

                pca = st.button("Click to see how PCA can speed up machine learning and to run a new regression model")

                if pca == True:

                    st.session\_state.pcasession = 'True'

                    with st.spinner('Please wait while we conduct a new linear regression using the principal components'):

                        my\_bar = st.progress(0)

                        time.sleep(10)

                        for percent\_complete in range(100):

                            time.sleep(0.01)

                            my\_bar.progress(percent\_complete + 1)

                        start\_time = time.time()

                        ### From Jupter - Principal component analysis continued

                        # Now use PCA to speed up machine learning

                        #from sklearn.model\_selection import train\_test\_split

                        # test\_size: what proportion of original data is used for test set

                        train\_X, test\_X, train\_y, test\_y = train\_test\_split(x, y, test\_size=1/4.0, random\_state=0)

                        # Scale the data

                        scaler = StandardScaler()

                        # Fit on training set only

                        scaler.fit(train\_X)

                        # Apply transform to both the training set and the test set.

                        train\_X = scaler.transform(train\_X)

                        test\_X = scaler.transform(test\_X)

                        # Choose minimum number of principal components such that 95% of the variance is retained

                        from sklearn.decomposition import PCA

                        # Make an instance of the model

                        pca = PCA(.95)

                        # Fit on training set

                        pca.fit(train\_X)

                        # Apply the mapping (transformation) to both the training set and the test set

                        train\_X = pca.transform(train\_X)

                        test\_X = pca.transform(test\_X)

                        # Apply model of choice, e.g. logistic regression - this will become dynamic in the app; choose model here

                        # Determine number of components

                        st.write("Number of useful components:")

                        st.write(pca.n\_components\_)

                        # Determine components

                        st.write("Component contributions:")

                        st.write(pca.components\_)

                        df3 = pd.DataFrame(pca.components\_)

                        # st.table(df3)

                        ### End

                        # tunedreg = st.button("Click to run a regression model with these components") # For brand

                        # if tunedreg == True and st.session\_state.pcasession == True:

                        ### From Jupyter - Linear regression

                        # Choose predicted variable - this will become dynamic in the app

                        y = df['BRAND'].sample(7500)

                        print(y.shape)

                        print(train\_X.shape)

                        # Define predictor variables

                        x = train\_X

                        x, y = np.array(x), np.array(y)

                        x = sm.add\_constant(x)

                        model = sm.OLS(y, x)

                        results = model.fit()

                        st.subheader("PCA regression results:")

                        st.write(results.summary())

                        st.write("")

                        st.write('\nPredicted response:', results.fittedvalues, sep='\n') # Or print('predicted response:', results.predict(x), sep='\n')

                        st.write("")

                        st.write("Conducting a new linear regression with principal components took ", time.time() - start\_time, "seconds to run")