st.subheader("Logistic regression")

with st.spinner('Please wait while we conduct the multinomial logistic regression analysis'):

    my\_bar = st.progress(0)

    time.sleep(10)

    for percent\_complete in range(100):

        time.sleep(0.1)

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

    start\_time = time.time()

    st.write("\*\*This model looks at the probability of consumers preferring particular brands (i.e. brand is the dependent categorical variable). With future ingestion of new data the scope of choosing additional categorical dependent variables may increase.\*\*")

    ### From Jupyter - 3. Multinomial logistic regression

    # Evaluate multinomial logistic regression model

    st.info("Basic assumptions that must be met for logistic regression include independence of errors, linearity in the logit for continuous variables, absence of multicollinearity, and lack of strongly influential outliers.")

    st.info("There are some fundamental differences between logistic, and linear, regression, in terms of necessary conditions on the data. First, logistic regression does not require a linear relationship between the dependent and independent variables.  Second, the error terms (residuals) do not need to be normally distributed.  Third, homoscedasticity is not required.  Finally, the dependent variable in logistic regression is not measured on an interval or ratio scale.")

    st.info("We should still have a large sample size (for normality), the dependent variable should be ordinal (or binary), and observations should be independent - i.e. repeated measurements or matched data is not used. Happily, these conditions are satisfied with our data.")

    st.warning("We can interpret the coefficients of the logistic regression as the probability of an observation fallinng into a particular class of dependent variable given changes in a set of inndependent, or input, variables. This means we can predict the potential outcome class for a relevant variable, if we know how other variables in the dataset change. This is what we do below, by predicting which brand a hypothetical consumer would prefer.")

    st.info("This will enable us to draw conclusions about the type of consumer that prefers each brand, based on experimentation.")

    link = f'[{"See more"}]({"https://www.statology.org/assumptions-of-logistic-regression/"})'

    st.markdown(link, unsafe\_allow\_html=True)

    st.write("Please set up a hypothetical consumer to determine their likely purchase habits")

    age = st.number\_input("Please enter consumer age")

    gender = st.radio("Please select consumer gender", options=["Male", "Female"])

    firstint = st.number\_input("How many days ago was the consumer's first interaction?")

    lastint = st.number\_input("How many days ago was the consumer's last interaction?")

    province = st.radio("Please select consumer province", ["Eastern Cape", "Free State", "Gauteng", "KwaZulu-Natal", "Limpopo", "Mpumalanga", "Northern Cape", "North West", "Western Cape"])

    email = st.radio("Has the consumer opted in for email notifications?", options=["Yes", "No"])

    sms = st.radio("Has the consumer opted in for sms notifications?", options=["Yes", "No"])

    push = st.radio("Has the consumer opted in for push notifications?", options=["Yes", "No"])

    if gender == "Male":

        gender = 1

    else:

        gender=2

    if province == "Eastern Cape":

        province=1

    if province == "Free State":

        province=2

    if province == "Gauteng":

        province=3

    if province == "KwaZulu-Natal":

        province=4

    if province == "Limpopo":

        province=5

    if province == "Mpumalanga":

        province=6

    if province == "Northern Cape":

        province=7

    if province == "North West":

        province=8

    if province == "Western Cape":

        province=9

    if email == "Yes":

        email = 1

    else:

        email = 2

    if sms == "Yes":

        sms = 1

    else:

        sms = 2

    if push == "Yes":

        push = 1

    else:

        push = 2

    # Define dataset

    # y = df['BRAND'].iloc[0:10000]

    # X = df.iloc[0:10000, 1:-1] #subsamping for efficiency and speed

    dflogreg = df.drop(columns=['BRANDNAME'])

    ylogreg = dflogreg['BRAND'].sample(10000)

    Xlogreg = dflogreg.drop(columns=["BRAND"]).sample(10000) #subsamping for efficiency and speed

    varnames = dflogreg.columns.values.tolist()

    Xlogreg, ylogreg = np.array(Xlogreg), np.array(ylogreg)

    # Define the multinomial logistic regression model

    model = LogisticRegression(multi\_class='multinomial', solver='lbfgs')

    # Define the model evaluation procedure

    cv = RepeatedStratifiedKFold(n\_splits=2, n\_repeats=3, random\_state=1)

    # Evaluate the model and collect the scores

    n\_scores = cross\_val\_score(model, Xlogreg, ylogreg, scoring='accuracy', cv=cv, n\_jobs=-1)

    # report the model performance

    st.write('Mean Accuracy: %.3f (%.3f)' % (mean(n\_scores), std(n\_scores)))

    # make a prediction with a multinomial logistic regression model

    # define dataset

    X\_trainlogreg, X\_testlogreg, y\_trainlogreg, y\_testlogreg = train\_test\_split(Xlogreg, ylogreg, test\_size=0.25, random\_state=42)

    # define the multinomial logistic regression model

    model = LogisticRegression(multi\_class='multinomial', solver='lbfgs')

    # fit the model on the training dataset

    model.fit(X\_trainlogreg, y\_trainlogreg)

    # define a single row of test data

    # row = X\_testlogreg[0,0:] # Previous approach

    row = [0, 0, firstint, lastint, 0, 0, 0, gender, 0, 0, 0, 0, province, 0, email, sms, push, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

    branddf = pd.read\_csv('Brandname encoding.csv', low\_memory=False)

    # predict the class label

    yhat = model.predict([row])

    # summarize the predicted class

    st.success('Predicted Class: %d' % yhat[0])

    brandpreference = branddf.loc[branddf.BRAND == yhat[0],'BRAND NAME'].values[0]

    st.success("Predicted brand consumer prefers: {}".format(brandpreference) )

    # st.write(branddf) # Put this back in once the data has been sorted out

    st.write("\*\*Predict a multinomial probability distribution for brands, based on whole dataset:\*\*")

    yhat = model.predict\_proba([row])

    # summarize the predicted probabilities

    # st.write('Predicted Probabilities: %s' % yhat[0])

    st.write('Predicted probabilities:')

    # data = {

    # "Variable": varnames,

    # "Pred Prob": yhat[0],

    # }

    # st.write(pd.DataFrame(data))

    #st.write(varnames)

    st.write(yhat[0].tolist())

    # plt.figure(figsize=(7, 3))

    fig, ax = plt.subplots(figsize=(10, 6))

    plt.plot(yhat[0].tolist())

    buf = BytesIO()

    fig.savefig(buf, format="png")

    # st.pyplot(fig)

    st.image(buf)

    # Tune penalty hyperparameter (next cell)

    ### End

    st.write("")

    st.write("Logistic regression took ", time.time() - start\_time, "seconds to run")

    st.write("")

tunelogreg = st.button("Tune penalty hyperparameter")

if tunelogreg == True:

    with st.spinner('Please wait while we tune the hyperparameters for the logistic regression 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 - tune multinomial logistic regression hyperparameters

        # define the multinomial logistic regression model with a default penalty

        LogisticRegression(multi\_class='multinomial', solver='lbfgs', penalty='l2', C=1.0)

        # get a list of models to evaluate

        def get\_models():

            models = dict()

            for p in [0.0, 0.0001, 0.001, 0.01, 0.1, 1.0]:

                # create name for model

                key = '%.4f' % p

                # turn off penalty in some cases

                if p == 0.0:

                    # no penalty in this case

                    models[key] = LogisticRegression(multi\_class='multinomial', solver='lbfgs', penalty='none')

                else:

                    models[key] = LogisticRegression(multi\_class='multinomial', solver='lbfgs', penalty='l2', C=p)

            return models

        # evaluate a give model using cross-validation

        def evaluate\_model(model, Xlogreg, ylogreg):

            # define the evaluation procedure

            cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)

            # evaluate the model

            scores = cross\_val\_score(model, Xlogreg, ylogreg, scoring='accuracy', cv=cv, n\_jobs=-1)

            return scores

        # get the models to evaluate

        models = get\_models()

        # evaluate the models and store results

        results, names = list(), list()

        for name, model in models.items():

            # evaluate the model and collect the scores

            scores = evaluate\_model(model, Xlogreg, ylogreg)

            # store the results

            results.append(scores)

            names.append(name)

            # summarize progress along the way

            st.write('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))

        # plot model performance for comparison

        fig, ax = plt.subplots(figsize=(10, 6))

        pyplot.boxplot(results, labels=names, showmeans=True)

        buf = BytesIO()

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

        # st.pyplot(fig)

        st.image(buf)

        st.write("Smaller C = larger penalty; our results show we need a larger penalty for better model performance")

        ### End

        st.write("")

        st.write("Tuning hyperparameters for the logistic regression took ", time.time() - start\_time, "seconds to run")

ANOVA

with st.spinner('Please wait while we run the ANOVA 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 - 5. ANOVA

    with st.expander("Explanation of ANOVA analysis"):

        st.write("Analysis of Variance (ANOVA) is used to test how different sample groups within a dataset compare to each other. In our case, it will be used to test either (a) how one (or a group) of brands compare to the overall dataset, or (b) how two different groups of brands compare to each other, with respect to a chosen variable.")

        st.write("Groups mean differences inferred by analyzing variances. ANOVA uses variance-based F test to check the group mean equality.")

        st.info("The null hypothesis that group means are equal is tested with an F-test for all groups, followed by post-hoc tests to see individual group differences.")

    with st.expander("Assumptions of ANOVA"):

        st.warning("1. Residuals (experimental error) are approximately normally distributed."

        "\n\n2. Homoscedasticity or Homogeneity of variances (variances are equal between treatment groups)."

        "\n\n3. Observations are sampled independently from each other."

        "\n\n4. The dependent variable should be continuous. If the dependent variable is ordinal or rank (e.g. Likert item data), it is more likely to violate the assumptions of normality and homogeneity of variances.")

    data = df.head(10000)

    brands = pd.Series(data['BRANDNAME'].astype(str).drop\_duplicates()).sort\_values().tolist() # Faster than .unique()

    brandsmod = ["All"] + brands

    st.markdown("\*\*Please choose brands of interest. If option 'All' is selected, or no option is selected, the analysis will be run for all brands.\*\*")

    st.info("Tip: it is instructive to compare analysis between brands for different variables.")

    # brandssel = []

    brandssel = st.multiselect("Please select brands of interest.", brandsmod, key="brands")

    st.markdown("\*\*Please choose second brands of interest.\*\*")

    st.info("Tip: it is instructive to compare analysis between brands for different variables.")

    # brandssel = []

    brandssel2 = st.multiselect("Please select second (comparison) brands of interest.", brands, key="brands2") # No "All" option for second one

    st.info("NB: If one brand is selected, ANOVA is run for the variable of choice below for that brand against all brands. If all brands are selected, then ANOVA is run for the chosen variable against brand choice for all brands. If two brands or brand groups are selected, the groups are compared to each other.")

    st.warning("If no interaction variable is chosen, ANOVA is run purely with the relationship between the chosen variable and brand choice. If an interaction variable is chosen, ANOVA is run incorporating both the direct effects of the variables on brand choice, but also the interaction between the two explanatory variables.")

    # dfanv = df.drop(['BRAND', 'BRANDNAME'], axis = 1, inplace = False)

    anovavars = list(df.columns)

    chosenvar = st.selectbox("Please select variable for ANOVA analysis", anovavars)

    if chosenvar == "BRANDNAME":

        chosenvar = "BRAND"

    chosenvarindex = df.columns.get\_loc(chosenvar)

    # groupvar = st.selectbox("Please choose a variable by which to group", anovavars)

    # if groupvar == "BRANDNAME":

    #     groupvar = "BRAND"

    # groupvarindex = df.columns.get\_loc(groupvar)

    interacvar = st.selectbox("Please select interaction variable for ANOVA analysis", anovavars)

    if interacvar == "BRANDNAME":

        interacvar = "BRAND"

    interacvarindex = df.columns.get\_loc(interacvar)

    print(brandssel)

    print(brandssel2)

    if brandssel != [] and brandssel2 == []:

        if "All" not in brandssel:

            data = data[data['BRANDNAME'].isin(brandssel)]

        if interacvar == []:

            #Create a boxplot

            fig = data.boxplot(chosenvar, by='BRAND', figsize=(18, 18)) # Dynamic to variables of choice

            fig.set\_xticklabels(fig.get\_xticklabels(),rotation=270)

            ctrl = data[chosenvar][data.BRAND == 'Castle Lager'] # Dynamic to brand and variables of choice

            grps = pd.unique(data.BRAND.values)

            d\_data = {grp:data[chosenvar][data.BRAND == grp] for grp in grps}

            k = len(pd.unique(data.BRAND))  # number of conditions

            N = len(data.values)  # conditions times participants

            n = data.groupby('BRAND').size() #Participants in each condition

            buf = BytesIO()

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

            st.image(buf)

            # Alternate boxplot

            ax = sns.boxplot(x=chosenvar, y='BRAND', data=data, color='#99c2a2')

            ax = sns.swarmplot(x=chosenvar, y="BRAND", data=data, color='#7d0013')

            buf = BytesIO()

            ax.figure.savefig(buf, format="png")

            st.image(buf)

            # ANOVA using statsmodels

            print(chosenvar)

            # Start of block comment

            # mod = ols(chosenvar + '~ BRAND',

            #                 data=data).fit()

            # aov\_table = sm.stats.anova\_lm(mod, typ=2)

            # # aovrm = AnovaRM(df, depvar='BRAND', subject=chosenvar, within=[chosenvar], aggregate\_func=mean)

            # # res = aovrm.fit()

            # aov\_table = sm.stats.anova\_lm(mod, typ=2)

            # st.write("ANOVA table of " + chosenvar + " appears below, grouped by brand. This shows the statistical significance of " + chosenvar + " on the choice of brands.")

            # st.table(aov\_table)

            # # Pairwise comparisons

            # print(data.head(5))

            # pair\_t = mod.t\_test\_pairwise('BRAND') # can add optional method = "sidak" or "bonferroni" here

            # st.write("The impact of " + chosenvar + " on the different brand choices appears below:")

            # st.write(pair\_t.result\_frame)

            # End of block comment

            res = stat()

            res.anova\_stat(df=data, res\_var='BRAND', anova\_model='BRAND ~ C(' + chosenvar + ')')

            st.write(res.anova\_summary)

            st.success("\*\*If the p-value (PR(>F)) is smaller than 0.05, we can reject the null hypothesis and conclude that the chosen group is significantly differently impacted by the chosen variables, compared to all brands.\*\*")

            with st.expander("Explaining the model pairwise comparison results"):

                st.info("If we conclude that there are significant differences between groups, we do not know which groups are different. To know the pairs of significant different treatments, we will perform multiple pairwise comparison (post hoc comparison) analysis for all unplanned comparison using Tukey's honestly significantly differenced (HSD) test."

                "\n\nThis will tell us which pairwise brand comparisons are significantly different, according to the below table.")

                st.table(branddf)

            res.tukey\_hsd(df=data, res\_var='BRAND', xfac\_var=chosenvar, anova\_model='BRAND ~ C(' + chosenvar + ')')

            st.write(res.tukey\_summary)

            st.success("In the pairwise comparisons, those with p-values less than 0.05 are significantly different to each other.")

            # res.anova\_std\_residuals are standardized residuals obtained from ANOVA (check above)

            sm.qqplot(res.anova\_std\_residuals, line='45')

            plt.xlabel("Theoretical Quantiles")

            plt.ylabel("Standardized Residuals")

            st.pyplot(plt)

            # histogram

            plt.hist(res.anova\_model\_out.resid, bins='auto', histtype='bar', ec='k')

            plt.ylim([0, 1.3\*max(res.anova\_model\_out.resid)])

            plt.xlabel("Residuals")

            plt.ylabel('Frequency')

            st.pyplot(plt)

            st.success("We can check to see if the standardized residuals lie around the 45-degree line, and that the distribution histogram is distributed normally, to see whether residuals are distributed normally, and our conclusions are valid, especially for larger sample sizes.")

            w, pvalue = stats.shapiro(data[chosenvar])

            st.info("The Shapiro-Wilk test can be used to check the normal distribution of residuals. Null hypothesis: data is drawn from the normal distribution.")

            st.write("Shapiro-Wilk test results: ", w, pvalue)

            if pvalue<0.05:

                st.error("The data indicates some processing may be required for more robust interpretation of results, as data is not normally distributed.")

            elif pvalue>0.05:

                st.success("We fail to reject the null hypothesis. Data approximates the normal distribution.")

            st.info("We can use Bartlett's test to check the Homogeneity of variances. Null hypothesis: samples from populations have equal variances.")

            w, pvalue = stats.bartlett(data[chosenvar], data['BRAND'])

            st.write("Bartlett test results: ", w, pvalue)

            if pvalue<0.05:

                st.success("The data indicates some processing may be required for more robust interpretation of results, as samples have unequal variances.")

            elif pvalue>0.05:

                st.success("We fail to reject the null hypothesis. Data samples exhibit equal variance.")

            # # if you have a stacked table, you can use bioinfokit v1.0.3 or later for the bartlett's test

            # res = stat()

            # res.bartlett(df=data, res\_var='BRAND', xfac\_var=df[chosenvar])

            # st.write(res.bartlett\_summary)

            res = stat()

            res.levene(df=data, res\_var='BRAND', xfac\_var=chosenvar)

            st.write(res.levene\_summary)

            ### End

        elif interacvar != []:

            fig = sns.boxplot(x=chosenvar, y="BRAND", hue=interacvar, data=data, palette="Set3")

            res = stat()

            res.anova\_stat(df=data, res\_var='BRAND', anova\_model='BRAND~C('+chosenvar+')+C('+interacvar+')+C('+chosenvar+'):C('+interacvar+')')

            st.write(res.anova\_summary)

            st.success("\*\*If the p-value is significant, we can conclude that either the explanatory variable, the interaction variable, or both through their interaction significantly impacts brand choice for consumers, in the choice between the two chosen groups.\*\*")

            st.info("We can also visualize the interaction plot (also called profile plot) for interaction effects:")

            fig = interaction\_plot(x=data.iloc[chosenvarindex], trace=data.iloc[interacvarindex], response=data['BRAND'])

            st.pyplot(fig)

            st.info("The interaction plot helps to visualize the means of the response of the two factors on one graph.")

            st.success("The interaction effect is significant between the Genotype and years if the lines are not parallel (roughly parallel factor lines indicate no interaction - additive model). This interaction is also called ordinal interaction if the lines do not cross each other.")

            st.info("For a more reliable conclusion of the interaction plot, it should be verified with the F-test for interaction. We do this through multiple pairwise comparisons to find out which groups of explanatory variables are most significant.")

            res = stat()

            # for main effect main variable

            res.tukey\_hsd(df=data, res\_var='BRAND', xfac\_var=chosenvar, anova\_model='BRAND~C('+chosenvar+')+C('+interacvar+')+C('+chosenvar+'):C('+interacvar+')')

            st.write("The effect of " + chosenvar + ":")

            st.write(res.tukey\_summary)

            # for main effect secondary variable

            res.tukey\_hsd(df=data, res\_var='BRAND', xfac\_var=interacvar, anova\_model='BRAND~C('+chosenvar+')+C('+interacvar+')+C('+chosenvar+'):C('+interacvar+')')

            st.write("The effect of " + interacvar + ":")

            st.write(res.tukey\_summary)

            # copy\_data = data.iloc[chosenvarindex] + data.iloc[interacvarindex] + data.iloc[0]

            # for interaction effect between chosen variable and interaction variable

            res.tukey\_hsd(df=data.head(10), res\_var='BRAND', xfac\_var=[chosenvar, interacvar], anova\_model='BRAND~C('+chosenvar+')+C('+interacvar+')+C('+chosenvar+'):C('+interacvar+')')

            st.write("The effect of interaction between " + chosenvar + " and " + interacvar + ":")

            st.write(res.tukey\_summary)

            st.info("We can use visual approaches, Bartlett's or Levene's, and the Shapiro-Wilk test, to validate the assumptions for homogeneity of variances and normal distribution of residuals.")

            # Shapiro-Wilk test

            import scipy.stats as stats

            w, pvalue = stats.shapiro(res.anova\_model\_out.resid)

            st.write("Results of the Shapiro-Wilk test: ", w, pvalue)

            if pvalue<0.05:

                st.error("The data indicates some processing may be required for more robust interpretation of results, as data is not normally distributed.")

            elif pvalue>0.05:

                st.success("We fail to reject the null hypothesis. Data approximates the normal distribution.")

            st.write("We should further look for the residual plots and histograms.")

             # res.anova\_std\_residuals are standardized residuals obtained from two-way ANOVA (check above)

            sm.qqplot(res.anova\_std\_residuals, line='45')

            plt.xlabel("Theoretical Quantiles")

            plt.ylabel("Standardized Residuals")

            st.pyplot(plt)

            # histogram

            plt.hist(res.anova\_model\_out.resid, bins='auto', histtype='bar', ec='k')

            plt.ylim([0, 1.3\*max(res.anova\_model\_out.resid)])

            plt.xlabel("Residuals")

            plt.ylabel('Frequency')

            st.pyplot(plt)

            st.success(" In the residual plot, if the standardized residuals lie around the 45-degree line, it suggests that the residuals are approximately normally distributed. Besides, the histogram shows whether residuals are normally distributed.")

            st.info("Note: The ANOVA model is remarkably robust to the violation of normality assumption, which means that it will have a non-significant effect on Type I error rate and p values will remain reliable as long as there are no outliers.")

            st.info("We will use Levene's test to check the assumption of homogeneity of variances")

            res = stat()

            res.levene(df=data, res\_var='BRAND', xfac\_var=[chosenvar, interacvar])

            st.write(res.levene\_summary)

            st.warning("If the p-value is non-significant (> 0.05), we fail to reject the null hypothesis and conclude that our samples have equal variances.")

    elif brandssel != [] and brandssel2 != []:

        if "All" not in brandssel:

            data = data[data['BRANDNAME'].isin(brands)]

        data2 = data[data['BRANDNAME'].isin(brandssel2)]

        data3 = data + data2

        if interacvar == []:

            #Create a boxplot

            fig = data.boxplot(chosenvar, by='BRAND', figsize=(18, 18)) # Dynamic to variables of choice

            fig.set\_xticklabels(fig.get\_xticklabels(),rotation=270)

            ctrl = data[chosenvar][data.BRAND == 'Castle Lager'] # Dynamic to brand and variables of choice

            grps = pd.unique(data.BRAND.values)

            d\_data = {grp:data[chosenvar][data.BRAND == grp] for grp in grps}

            k = len(pd.unique(data.BRAND))  # number of conditions

            N = len(data.values)  # conditions times participants

            n = data.groupby('BRAND').size()[0] #Participants in each condition

            buf = BytesIO()

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

            st.image(buf)

            # ANOVA using statsmodels

            print(chosenvar)

            # Start of block comment

            # mod = ols(chosenvar + '~ BRAND',

            #                 data=data).fit()

            # aov\_table = sm.stats.anova\_lm(mod, typ=2)

            # # aovrm = AnovaRM(df, depvar='BRAND', subject=chosenvar, within=[chosenvar], aggregate\_func=mean)

            # # res = aovrm.fit()

            # aov\_table = sm.stats.anova\_lm(mod, typ=2)

            # st.write("ANOVA table of " + chosenvar + " appears below, grouped by brand. This shows the statistical significance of " + chosenvar + " on the choice of brands.")

            # st.table(aov\_table)

            # # Pairwise comparisons

            # print(data.head(5))

            # pair\_t = mod.t\_test\_pairwise('BRAND') # can add optional method = "sidak" or "bonferroni" here

            # st.write("The impact of " + chosenvar + " on the different brand choices appears below:")

            # st.write(pair\_t.result\_frame)

            # End of block comment

            res = stat()

            res.anova\_stat(df=data3, res\_var='BRAND', anova\_model='BRAND ~ C(' + chosenvar + ')')

            st.write(res.anova\_summary)

            st.success("\*\*If the p-value (PR(>F)) is smaller than 0.05, we can reject the null hypothesis and conclude that the chosen group is significantly differently impacted by the chosen variables, compared to all brands.\*\*")

            with st.expander("Explaining the model pairwise comparison results"):

                st.info("If we conclude that there are significant differences between groups, we do not know which groups are different. To know the pairs of significant different treatments, we will perform multiple pairwise comparison (post hoc comparison) analysis for all unplanned comparison using Tukey's honestly significantly differenced (HSD) test."

                "\n\nThis will tell us which pairwise brand comparisons are significantly different, according to the below table.")

                st.table(branddf)

            res.tukey\_hsd(df=data3, res\_var='BRAND', xfac\_var=chosenvar, anova\_model='BRAND ~ C(' + chosenvar + ')')

            st.write(res.tukey\_summary)

            st.success("In the pairwise comparisons, those with p-values less than 0.05 are significantly different to each other.")

            # res.anova\_std\_residuals are standardized residuals obtained from ANOVA (check above)

            sm.qqplot(res.anova\_std\_residuals, line='45')

            plt.xlabel("Theoretical Quantiles")

            plt.ylabel("Standardized Residuals")

            st.pyplot(plt)

            # histogram

            plt.hist(res.anova\_model\_out.resid, bins='auto', histtype='bar', ec='k')

            plt.ylim([0, 1.3\*max(res.anova\_model\_out.resid)])

            plt.xlabel("Residuals")

            plt.ylabel('Frequency')

            st.pyplot(plt)

            st.success("We can check to see if the standardized residuals lie around the 45-degree line, and that the distribution histogram is distributed normally, to see whether residuals are distributed normally, and our conclusions are valid, especially for larger sample sizes.")

            w, pvalue = stats.shapiro(data3[chosenvar])

            st.info("The Shapiro-Wilk test can be used to check the normal distribution of residuals. Null hypothesis: data is drawn from the normal distribution.")

            st.write("Shapiro-Wilk test results: ", w, pvalue)

            if pvalue<0.05:

                st.error("The data indicates some processing may be required for more robust interpretation of results, as data is not normally distributed.")

            elif pvalue>0.05:

                st.success("We fail to reject the null hypothesis. Data approximates the normal distribution.")

            st.info("We can use Bartlett's test to check the Homogeneity of variances. Null hypothesis: samples from populations have equal variances.")

            w, pvalue = stats.bartlett(data3[chosenvar], data3['BRAND'])

            st.write("Bartlett test results: ", w, pvalue)

            if pvalue<0.05:

                st.success("The data indicates some processing may be required for more robust interpretation of results, as samples have unequal variances.")

            elif pvalue>0.05:

                st.success("We fail to reject the null hypothesis. Data samples exhibit equal variance.")

            # # if you have a stacked table, you can use bioinfokit v1.0.3 or later for the bartlett's test

            # res = stat()

            # res.bartlett(df=data, res\_var='BRAND', xfac\_var=df[chosenvar])

            # st.write(res.bartlett\_summary)

            # res = stat()

            # res.levene(df=data, res\_var='BRAND', xfac\_var=df[chosenvar])

            # st.write(res.levene\_summary)

        elif interacvar != []:

            fig = sns.boxplot(x=chosenvar, y="BRAND", hue=interacvar, data=data, palette="Set3")

            res = stat()

            res.anova\_stat(df=data, res\_var='BRAND', anova\_model='BRAND~C('+chosenvar+')+C('+interacvar+')+C('+chosenvar+'):C('+interacvar+')')

            st.write(res.anova\_summary)

            st.success("\*\*If the p-value is significant, we can conclude that either the explanatory variable, the interaction variable, or both through their interaction significantly impacts brand choice for consumers, in the choice between the two chosen groups.\*\*")

            st.info("We can also visualize the interaction plot (also called profile plot) for interaction effects:")

            fig = interaction\_plot(x=data.iloc[chosenvarindex], trace=data.iloc[interacvarindex], response=data['BRAND'])

            st.pyplot(fig)

            st.info("The interaction plot helps to visualize the means of the response of the two factors on one graph.")

            st.success("The interaction effect is significant between the Genotype and years if the lines are not parallel (roughly parallel factor lines indicate no interaction - additive model). This interaction is also called ordinal interaction if the lines do not cross each other.")

            st.info("For a more reliable conclusion of the interaction plot, it should be verified with the F test for interaction. We do this through multiple pairwise comparisons to find out which groups of explanatory variables are most significant.")

            res = stat()

            # for main effect main variable

            res.tukey\_hsd(df=data, res\_var='BRAND', xfac\_var=chosenvar, anova\_model='BRAND~C('+chosenvar+')+C('+interacvar+')+C('+chosenvar+'):C('+interacvar+')')

            st.write("The effect of " + chosenvar + ":")

            st.write(res.tukey\_summary)

            # for main effect secondary variable

            res.tukey\_hsd(df=data, res\_var='BRAND', xfac\_var=interacvar, anova\_model='BRAND~C('+chosenvar+')+C('+interacvar+')+C('+chosenvar+'):C('+interacvar+')')

            st.write("The effect of " + interacvar + ":")

            st.write(res.tukey\_summary)

            # for interaction effect between chosen variable and interaction variable

            res.tukey\_hsd(df=data, res\_var='BRAND', xfac\_var=[chosenvar, interacvar], anova\_model='BRAND~C('+chosenvar+')+C('+interacvar+')+C('+chosenvar+'):C('+interacvar+')')

            st.write("The effect of interaction between " + chosenvar + " and " + interacvar + ":")

            st.write(res.tukey\_summary)

            st.info("We can use visual approaches, Bartlett's or Levene's, and the Shapiro-Wilk test, to validate the assumptions for homogeneity of variances and normal distribution of residuals.")

            # Shapiro-Wilk test

            import scipy.stats as stats

            w, pvalue = stats.shapiro(res.anova\_model\_out.resid)

            st.write("Results of the Shapiro-Wilk test: ", w, pvalue)

            if pvalue<0.05:

                st.error("The data indicates some processing may be required for more robust interpretation of results, as data is not normally distributed.")

            elif pvalue>0.05:

                st.success("We fail to reject the null hypothesis. Data approximates the normal distribution.")

            st.write("We should further look for the residual plots and histograms.")

             # res.anova\_std\_residuals are standardized residuals obtained from two-way ANOVA (check above)

            sm.qqplot(res.anova\_std\_residuals, line='45')

            plt.xlabel("Theoretical Quantiles")

            plt.ylabel("Standardized Residuals")

            st.pyplot(plt)

            # histogram

            plt.hist(res.anova\_model\_out.resid, bins='auto', histtype='bar', ec='k')

            plt.ylim([0, 1.3\*max(res.anova\_model\_out.resid)])

            plt.xlabel("Residuals")

            plt.ylabel('Frequency')

            st.pyplot(plt)

            st.success(" In the residual plot, if the standardized residuals lie around the 45-degree line, it suggests that the residuals are approximately normally distributed. Besides, the histogram shows whether residuals are normally distributed.")

            st.info("Note: The ANOVA model is remarkably robust to the violation of normality assumption, which means that it will have a non-significant effect on Type I error rate and p values will remain reliable as long as there are no outliers.")

            st.info("We will use Levene's test to check the assumption of homogeneity of variances")

            res = stat()

            res.levene(df=data, res\_var='BRAND', xfac\_var=[chosenvar, interacvar])

            st.write(res.levene\_summary)

            st.warning("If the p-value is non-significant (> 0.05), we fail to reject the null hypothesis and conclude that our samples have equal variances.")

        ### End

        st.write("")

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

PCA

import hydralit as hy

from numpy.core.fromnumeric import var

import streamlit

import streamlit as st

import sys

from streamlit import cli as stcli

from PIL import Image

from functions import \*

import streamlit.components.v1 as components

import pandas as pd

from st\_clickable\_images import clickable\_images

import numpy as np

import statsmodels.api as sm

from numpy import mean

from numpy import std

from sklearn.datasets import make\_classification

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import RepeatedStratifiedKFold

from sklearn.linear\_model import LogisticRegression

from matplotlib import pyplot

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

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import silhouette\_score

from scipy.spatial.distance import cdist

import seaborn as sns

from io import BytesIO

from statsmodels.formula.api import ols

from streamlit.state.session\_state import SessionState

import tkinter

import matplotlib

# matplotlib.use('TkAgg')

# matplotlib.use('Agg')

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import accuracy\_score

from mpl\_toolkits.mplot3d import Axes3D

from matplotlib import cm

from matplotlib.ticker import LinearLocator, FormatStrFormatter

from sklearn.tree import DecisionTreeRegressor, plot\_tree

import sklearn

from sklearn.datasets import make\_classification

from sklearn.ensemble import RandomForestClassifier

from sklearn.datasets import make\_regression

from sklearn.ensemble import RandomForestRegressor

from sklearn.model\_selection import RepeatedKFold

import time

from scipy.stats import pearsonr

from scipy.stats import spearmanr

import dtale

from dtale.views import startup

from dtale.app import get\_instance

import webbrowser

import dtale.global\_state as global\_state

import dtale.app as dtale\_app

from matplotlib.pyplot import axis, hist

from scipy import stats as stats

from bioinfokit.analys import stat

from statsmodels.stats.anova import AnovaRM

import statsmodels.api as sm

from statsmodels.graphics.factorplots import interaction\_plot

from sklearn.decomposition import PCA

title = '<p style="font-family:sans-serif; color:red; font-size: 39px; text-align: center;"><b>Code testing environment</b></p>'

st.markdown(title, unsafe\_allow\_html=True)

### From Jupyter - 0. Prepare the data

# Read data

df = pd.read\_csv('allrecordsohe.csv', low\_memory=False)

df2 = pd.read\_csv('allrecords.csv', low\_memory=False)

branddf = pd.read\_csv('Brandname encoding.csv', low\_memory=False)

# Check for empty data

df.isnull().sum()

df2.isnull().sum()

# Remove NaN

nr\_samples\_before = df.shape[0]

df = df.fillna(0)

print('Removed %s samples' % (nr\_samples\_before - df.shape[0]))

nr\_samples\_before = df2.shape[0]

df2 = df2.fillna(0)

print('Removed %s samples' % (nr\_samples\_before - df2.shape[0]))

# Drop irrelevant variables

df.drop(['TD\_ID', 'KRUX\_ID', 'TAP\_IT\_ID', 'GOOGLE\_CLIENT\_ID'], axis=1, inplace=True)

df2.drop(['TD\_ID', 'KRUX\_ID', 'TAP\_IT\_ID', 'GOOGLE\_CLIENT\_ID'], axis=1, inplace=True)

# df = df.reset\_index()

# df2 = df2.reset\_index()

### End

### Enter code to test here

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

    my\_bar = st.progress(0)

    time.sleep(1)

    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(1)

#         for percent\_complete in range(100):

#             time.sleep(0.1)

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

#         start\_time = time.time()

    st.write("We can now see how PCA can speed up machine learning to run a new regression model")

    ### 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

    # 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

    train\_X, y = np.array(train\_X), np.array(y)

    train\_X = sm.add\_constant(train\_X)

    model = sm.OLS(y, train\_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")