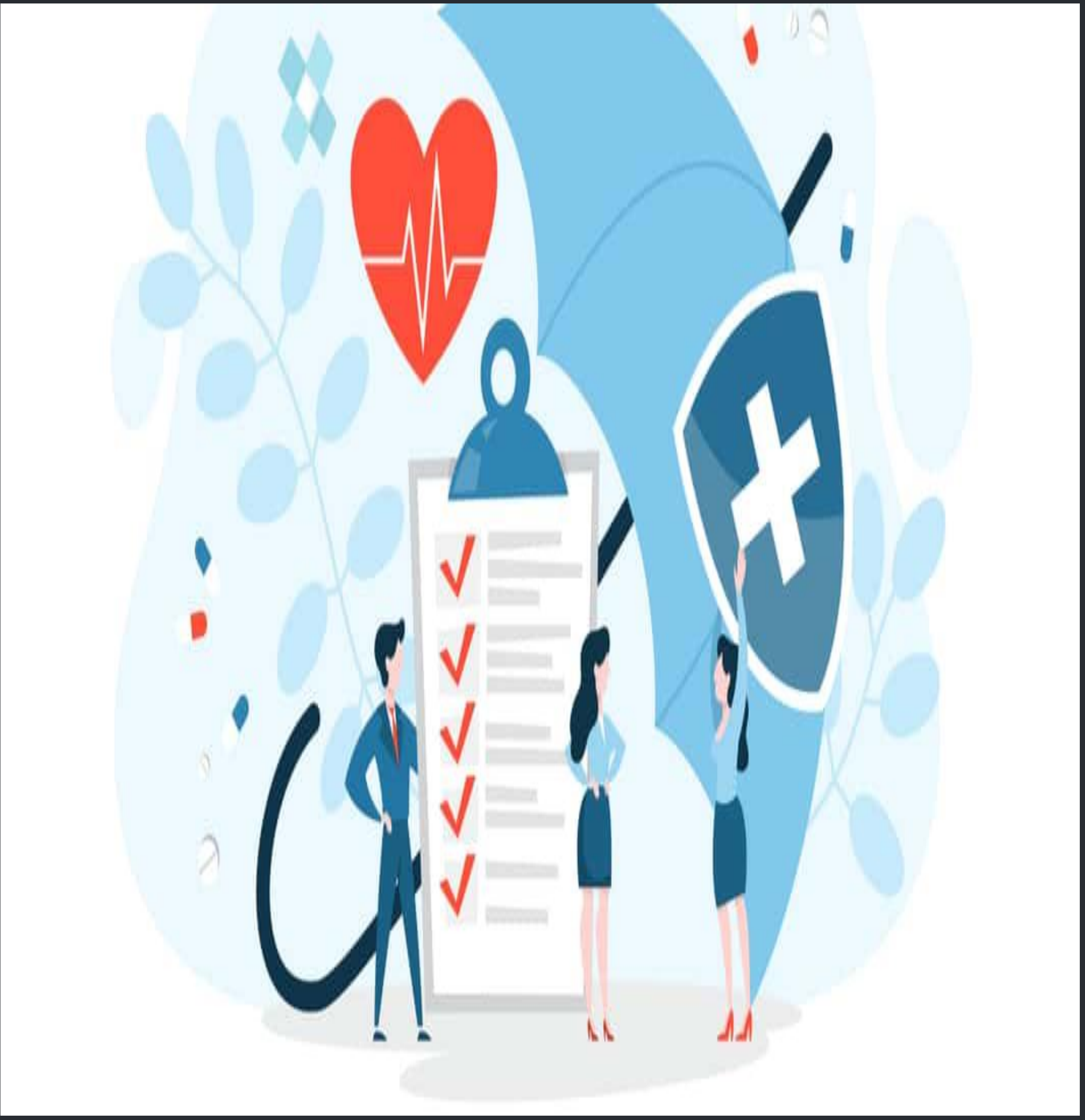


Public Health Awareness Campaign

Lets get started with our presentation on the Public Health Awareness Campaign. In this campaign, we aim to raise awareness and educate the public about various important health issues. And Examine the success of future camapigns based on the analysis of mental health prediction dataset. Come on Let's dive in and explore the contents and topics covered in this presentation.

WELCOME

To Our Project for IBM Data Analytics with Cognos for Public Health Awareness Campaign

Data Extraction in Notebook

1 **Web Scraping**

Explore web scraping techniques to extract relevant data for our campaign from various online sources.

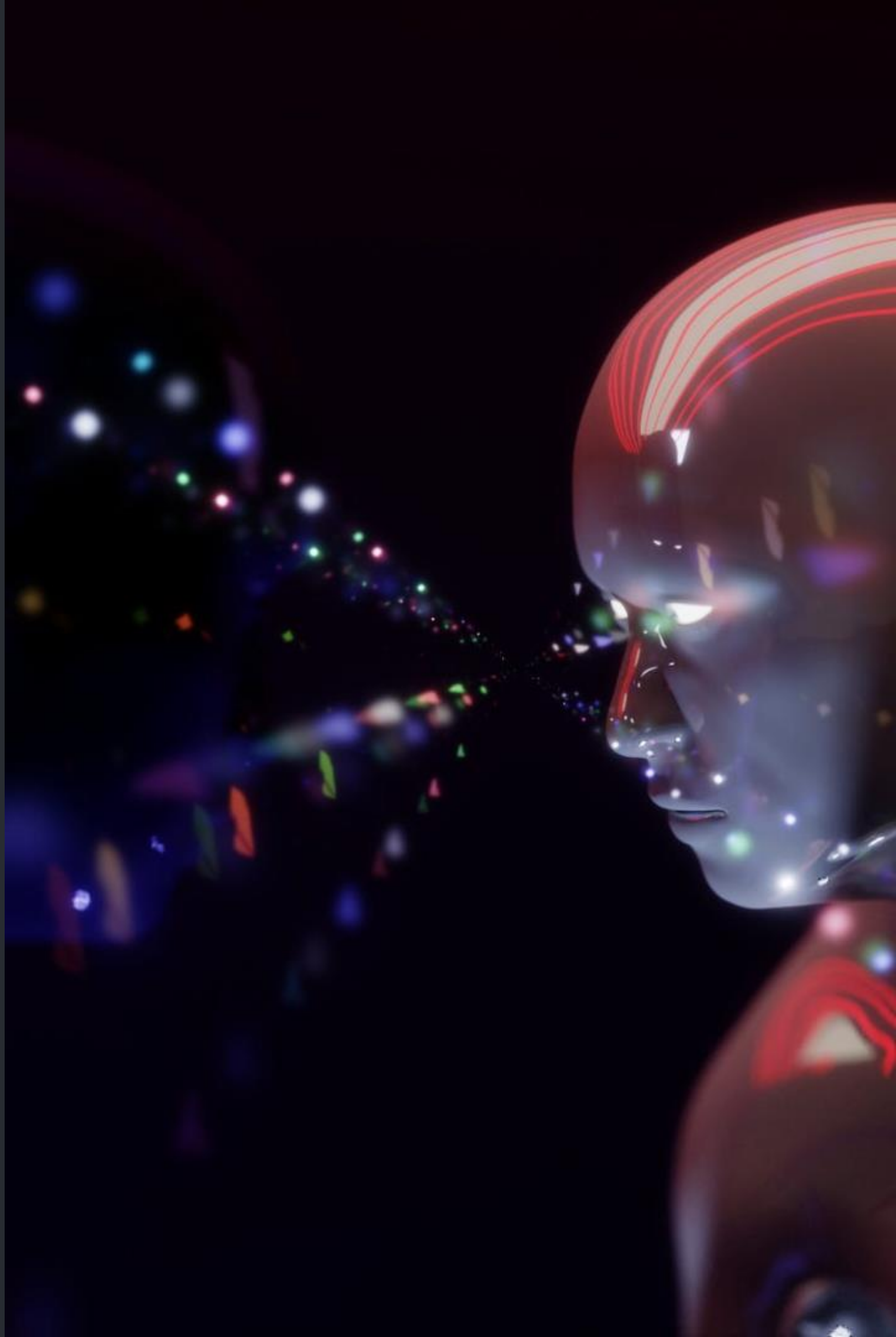
2 **API Integration**

Learn how to integrate APIs to fetch real-time data and make our campaign more informative and up-to-date.

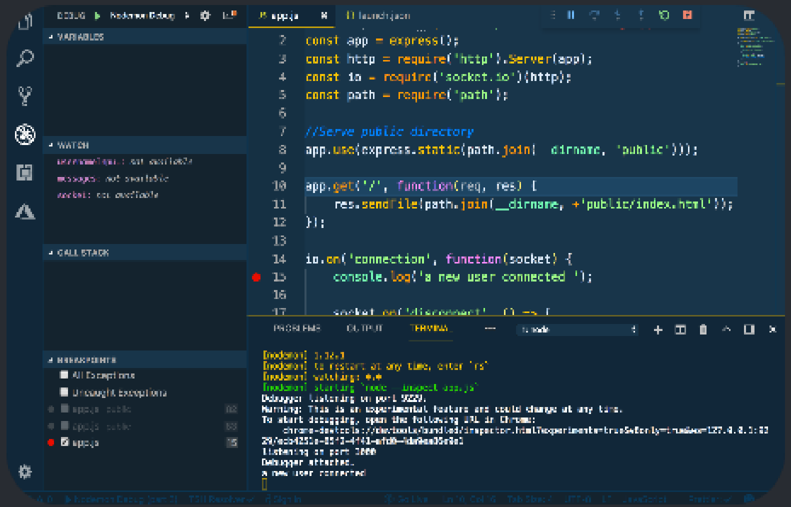
3 **Data Cleaning and Preprocessing**

Understand the importance of data cleaning and preprocessing to ensure the accuracy and reliability of the extracted data.

Ways to implement Machine Learning Algorithms to predict the success of future campaigns based on historical data of mental health prediction dataset.



Jupyter Notebook Code Implementation



Introduction to Jupyter Notebook

Learn how to set up and navigate Jupyter Notebook for implementing our campaign strategies.

Coding in Jupyter Notebook

Understand the basics of coding in Jupyter Notebook to effectively execute our campaign tasks.

Debugging Techniques

Discover useful debugging techniques to troubleshoot any issues encountered during the code implementation process.

Procedure

Jupyter Notebook Code Implementation

Get hands-on experience with Jupyter Notebook to implement different aspects of the campaign.

Data Extraction in Notebook

Learn how to effectively extract data using Jupyter Notebook for our campaign.

Machine Learning Algorithms

Discover the power of machine learning algorithms like Random Forest, K Classifier, CNN, KNN, and Gradient Descent for enhancing our campaign.

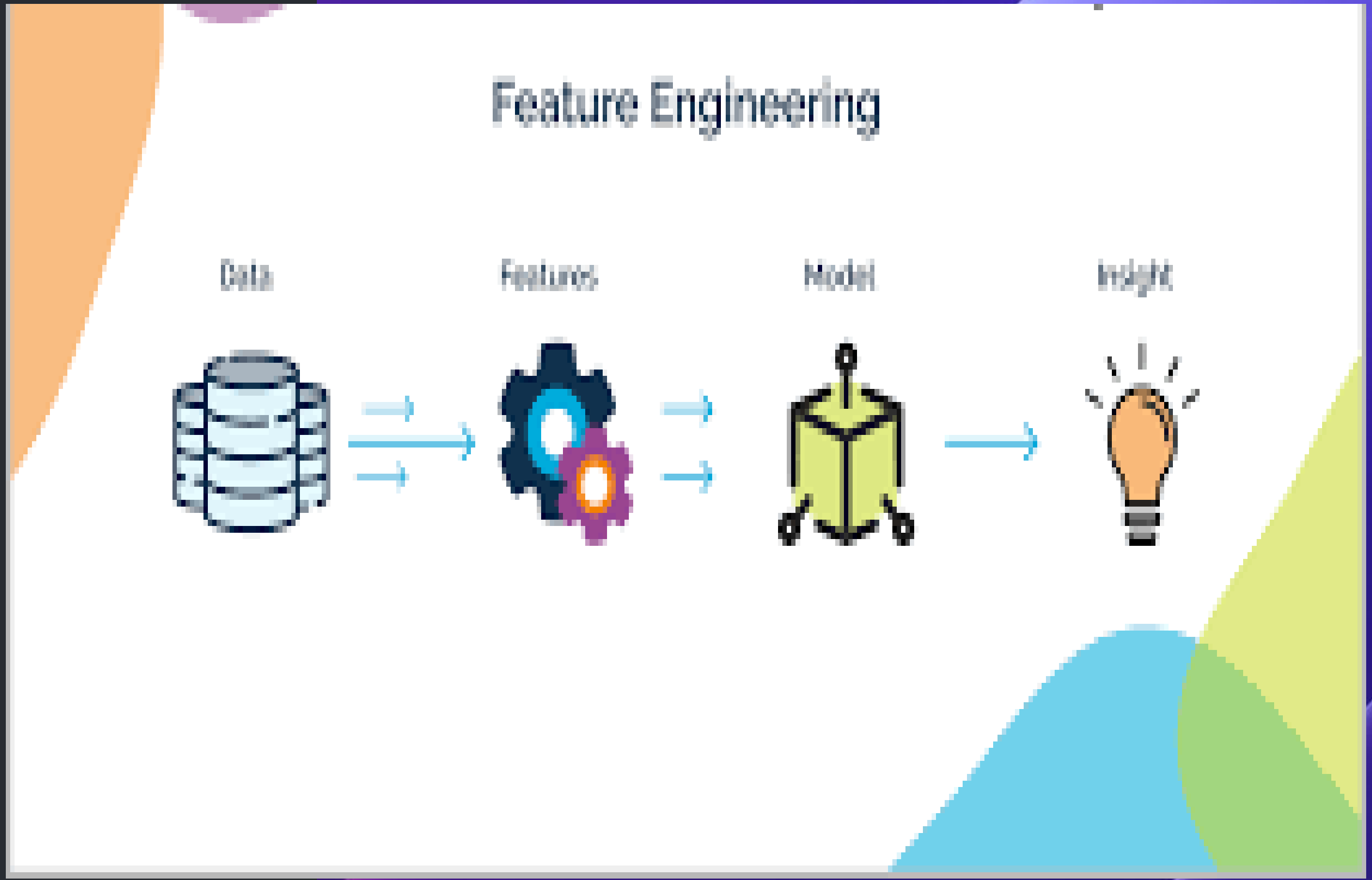
Data Collection and Preparation:

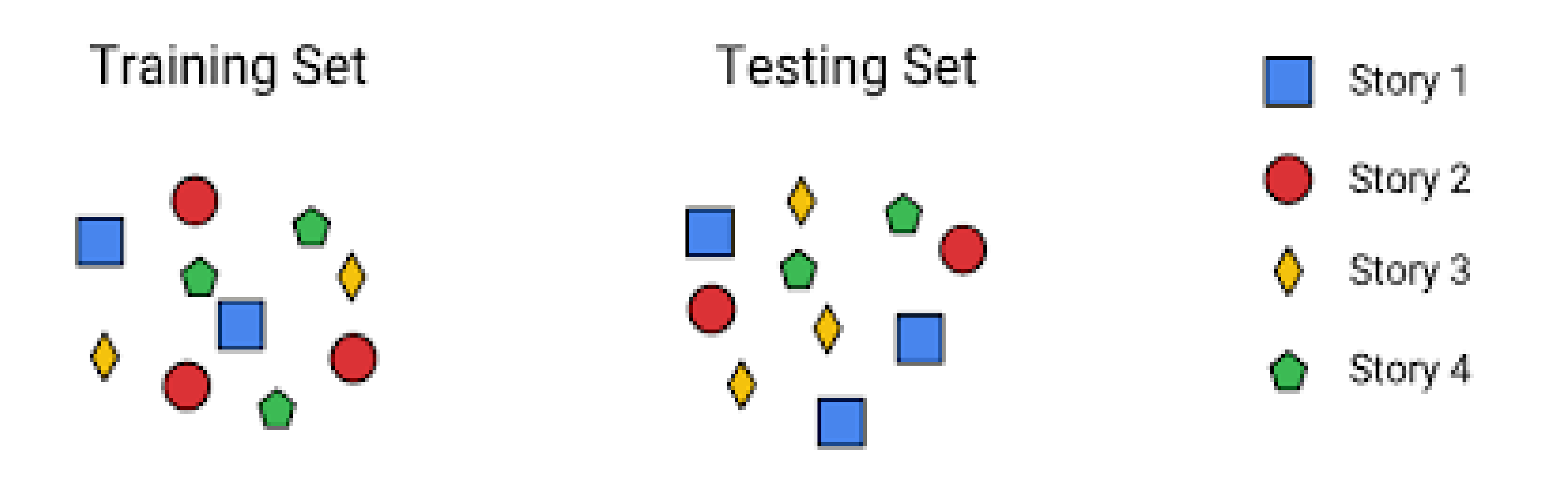
- Gather historical data on past campaigns. This data should include various features like campaign type, target audience, messaging, channels used, timing, and the outcome (e.g., conversion rate, ROI).

- Clean and preprocess the data to handle missing values, outliers, and ensure data consistency.

Feature Engineering:

- Create relevant features from the raw data that can help the machine learning model make accurate predictions. For example, you might create features like day of the week, seasonality, or customer segmentation based on demographics.





Data Splitting:

- Split your dataset into training, validation, and test sets. The training set is used to train the machine learning model, the validation set is used to tune hyperparameters and evaluate model performance during development, and the test set is reserved for final model evaluation.

Model Selection:

- Choose an appropriate machine learning algorithm for your prediction task. Common algorithms for classification and regression tasks include decision trees, random forests, logistic regression, gradient boosting, and neural networks.

Model Training:

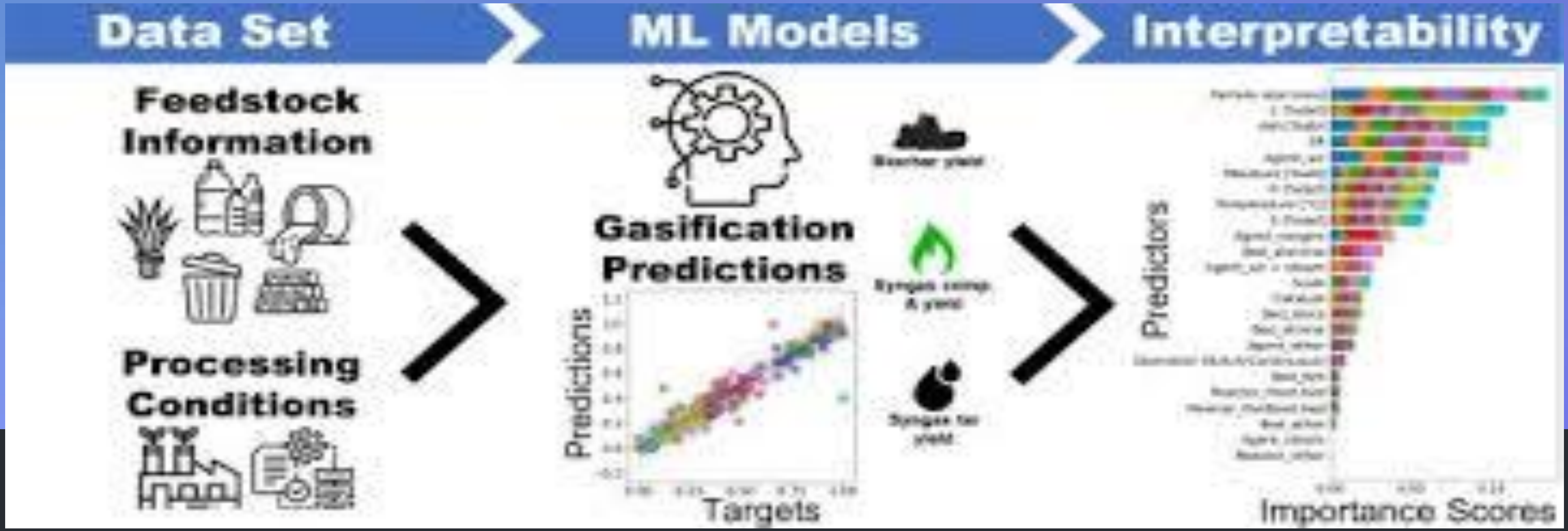
- Train your selected machine learning model on the training data using appropriate hyperparameters. Fine-tune the model to achieve the best performance on the validation set. This process may involve cross-validation and hyperparameter tuning.

Evaluation Metrics :

- Select appropriate evaluation metrics for your campaign prediction task. Common metrics include accuracy, precision, recall, F1-score, mean absolute error (MAE), mean squared error (MSE), or root mean squared error (RMSE).

Monitoring and Maintenance:

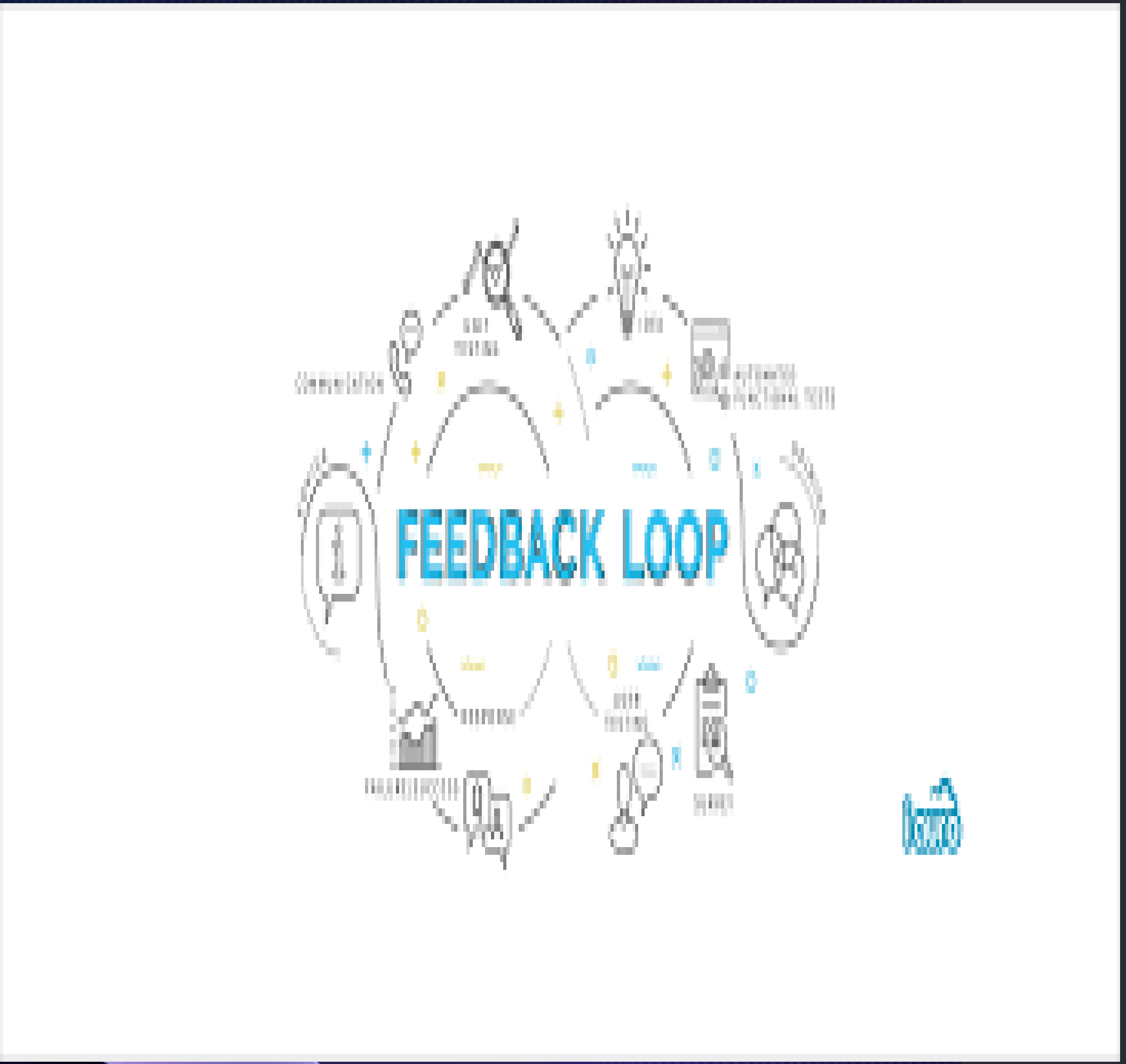
- Continuously monitor the model's performance in a production environment and retrain it periodically with new data to keep it up to date.



Model Interpretability:

Depending on the algorithm chosen, consider methods for interpreting the model's predictions. This is important for understanding which factors are driving campaign success prediction**Deployment**:

- Once you have a trained and validated model, deploy it into your operational environment. This could involve integrating it into your campaign management software or using it to make predictions through an API.



Feedback Loop:

- Collect feedback on the model's predictions and incorporate this feedback into future campaigns. This iterative process can help improve the accuracy of your predictions over time.

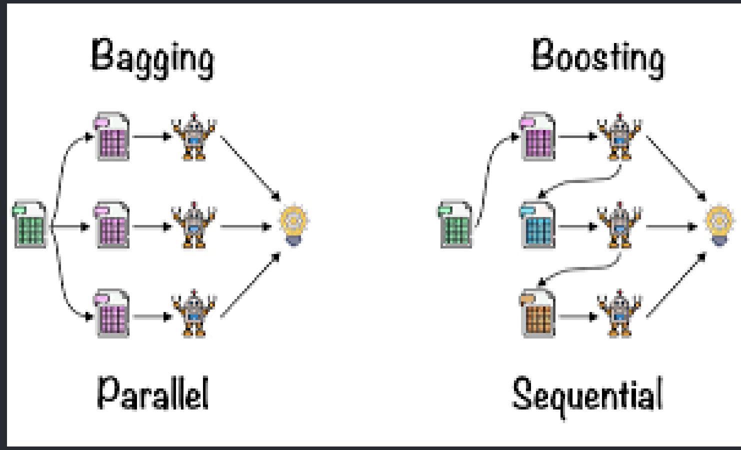
Ethical Considerations:

Ensure that your predictive model is developed and deployed in an ethical manner, considering issues like bias, fairness, and privacy.

Remember that the success of your machine learning model depends not only on the algorithm but also on the quality of the data, feature engineering, and domain knowledge. Regularly reevaluating and updating your model is crucial to maintain its predictive accuracy as market dynamics change.

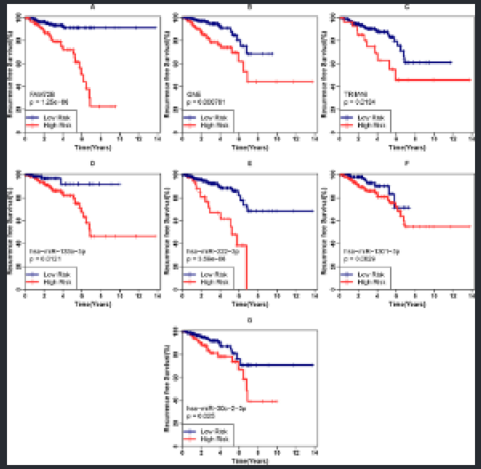
Additionally, it's advisable to work with data scientists or machine learning experts to implement and maintain such predictive models effectively.

Machine Learning Algorithms



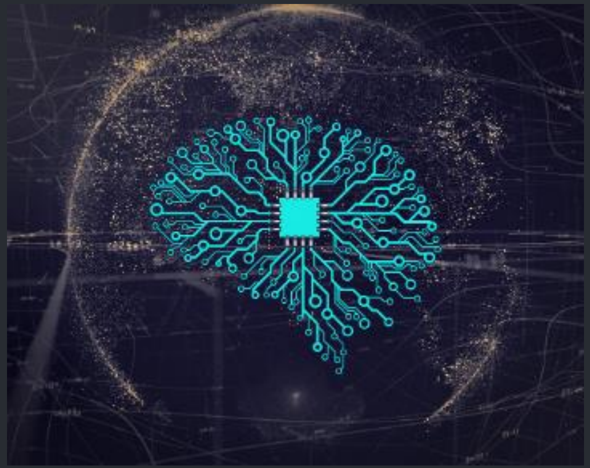
Random Forest

Explore the Random Forest algorithm and its application in predicting health trends and outcomes.



K Classifier

Dive into the K Classifier algorithm and understand its role in analyzing health-related data for our campaign.

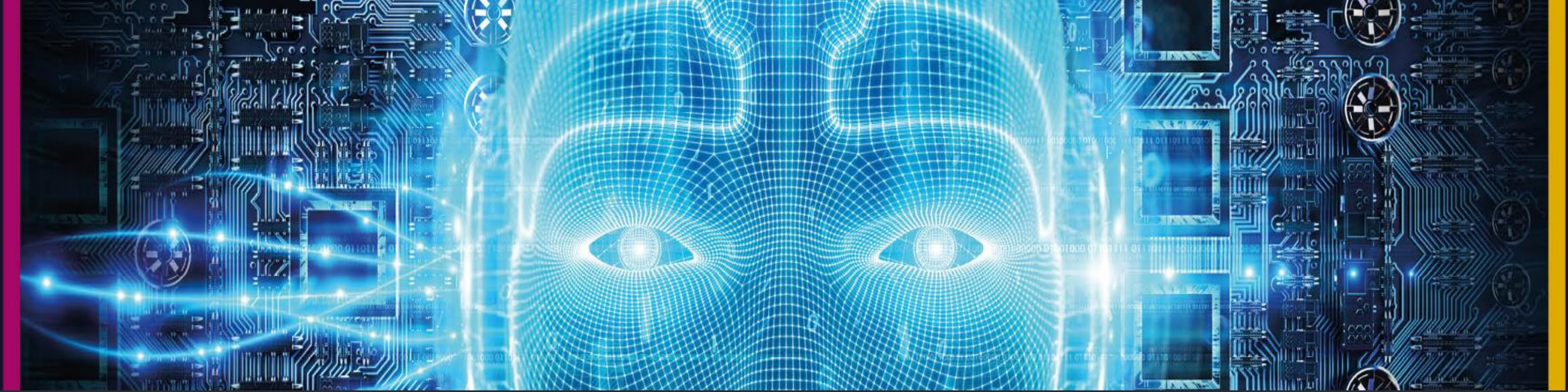


CNN (Convolutional Neural Network)

Discover the power of CNN in image analysis for our campaign, focusing on health-related images.

KNN (K-Nearest Neighbors)

Learn how the KNN algorithm can be utilized to identify patterns and make predictions in public health data.



Machine learning Code Implementation for prediction of success rate in future

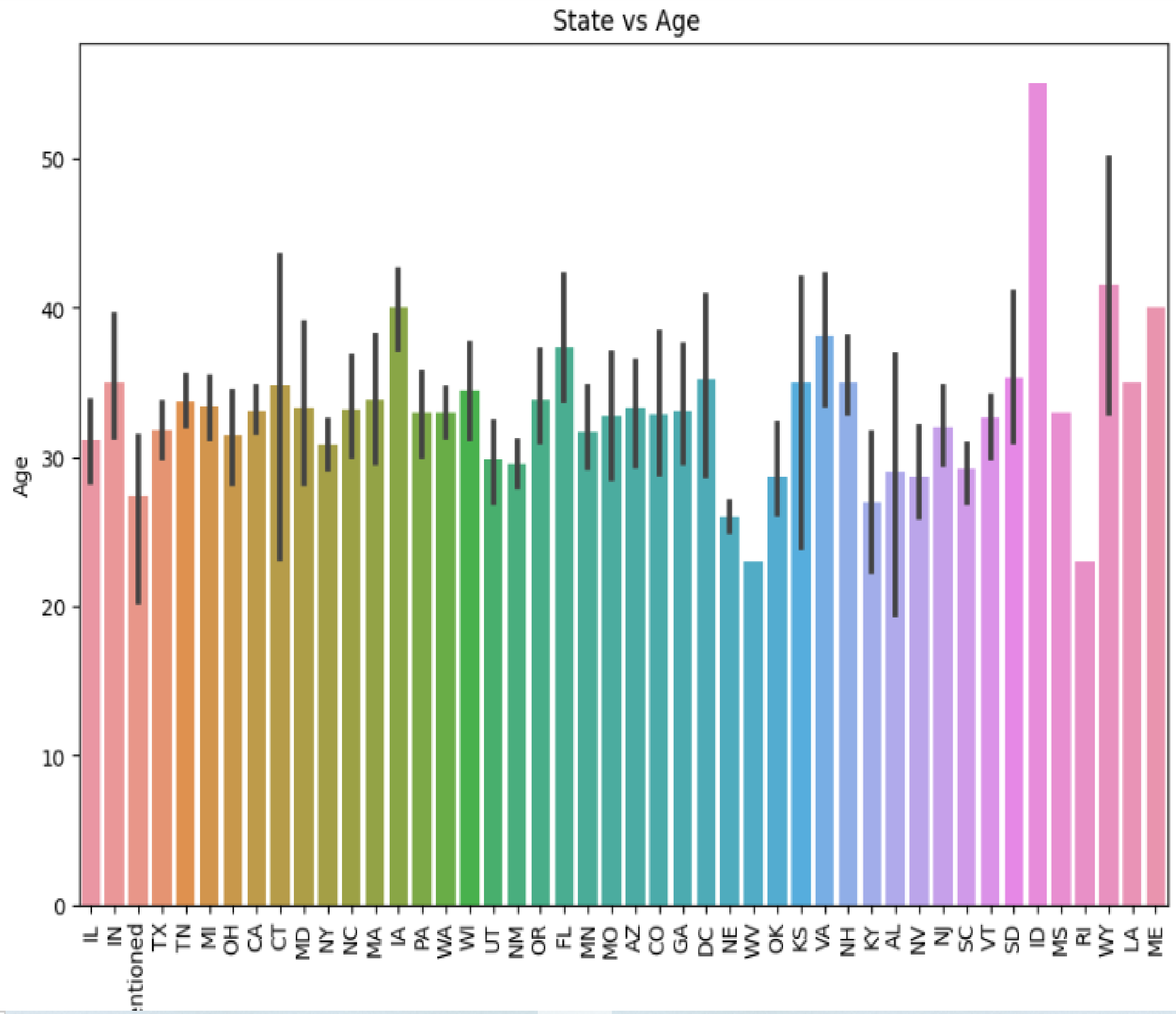
```
[5]: plt.figure(figsize = (10,7))
     sns.barplot(data = df,x='state',y='Age')
     plt.xticks(rotation=90)
     plt.title(" State vs Age")
     plt.show()
```

Jupyter Untitled1 Last Checkpoint: 12 days ago

File Edit View Run Kernel Settings Help

Code

JupyterLab



State Vs Age

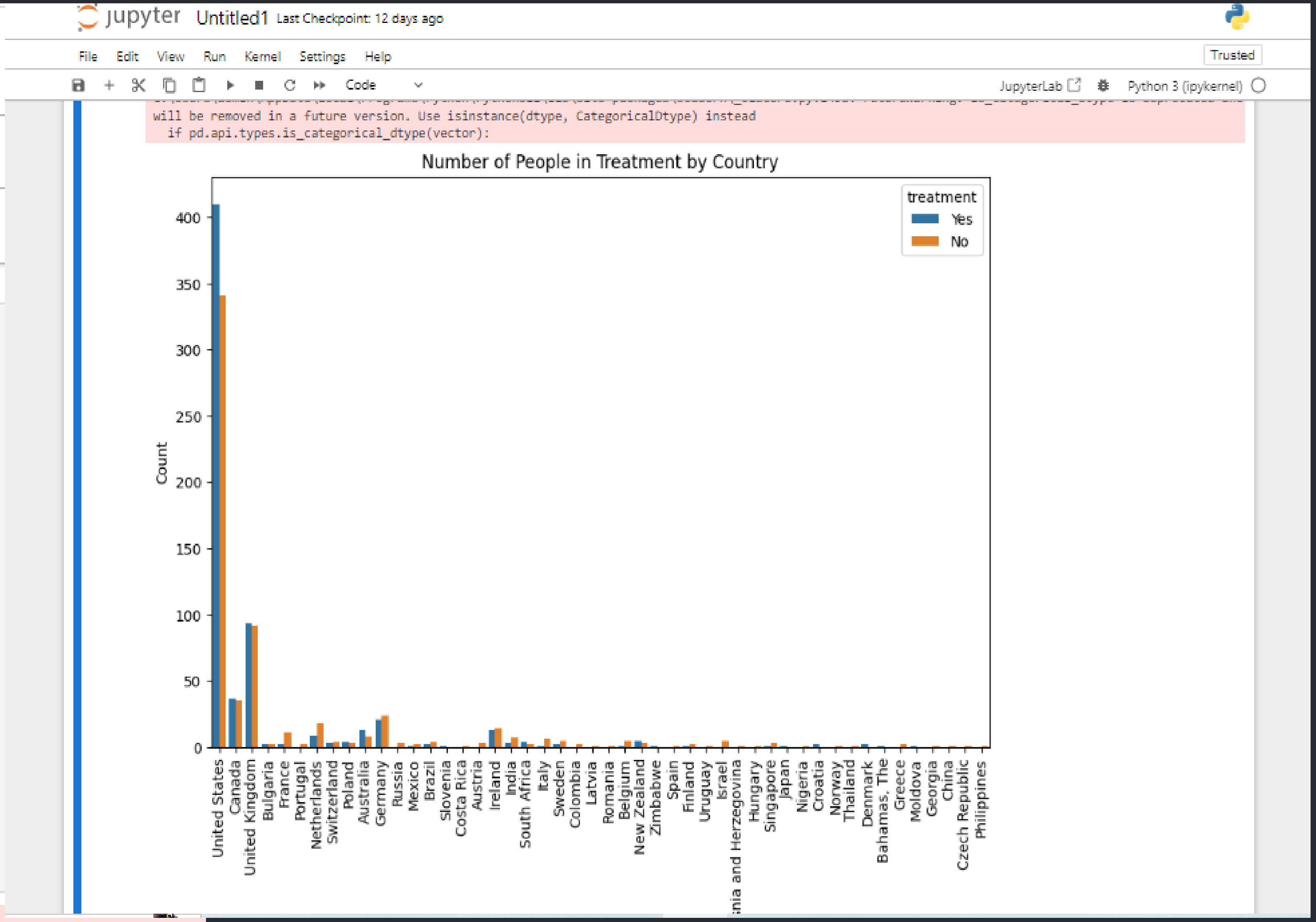
Number of people in Treatment by Country

Untitled1 Last Checkpoint: 12 days ago

File Edit View Run Kernel Settings Help

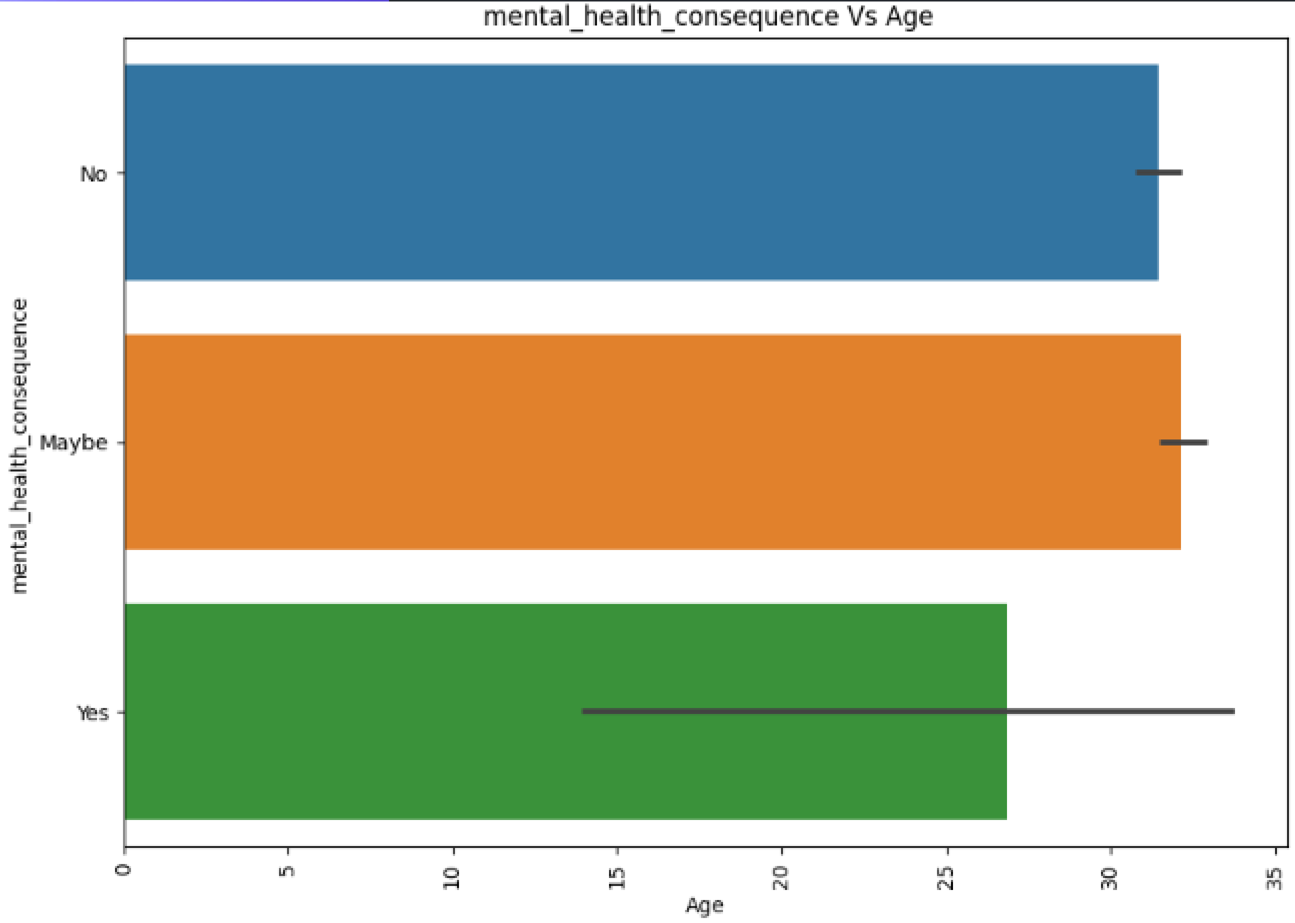
[4]:

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
df = pd.read_csv('C:\\Users\\admin\\Desktop\\survey.csv')
plt.figure(figsize=(10, 7))
sns.countplot(data=df, x='Country', hue='treatment')
plt.xticks(rotation=90)
plt.title('Number of People in Treatment by Country')
plt.xlabel('Country')
plt.ylabel('Count')
plt.show()
```



Mental health consequences Vs Age

```
[6]: plt.figure(figsize = (10,7))
     sns.barplot(data = df,x='Age',y='mental_health_consequence')
     plt.xticks(rotation=90)
     plt.title(" mental_health_consequence Vs Age")
     plt.show()
```



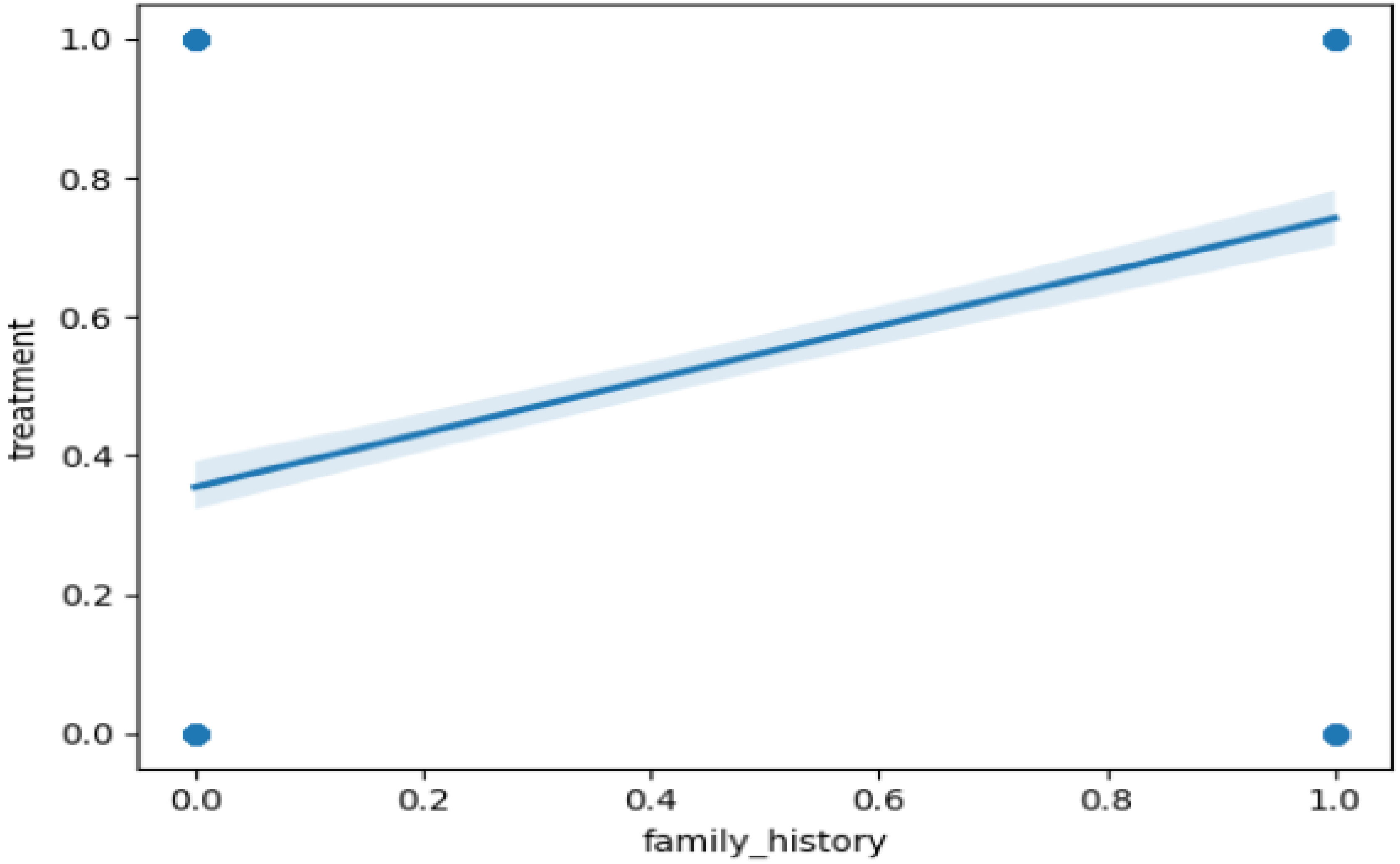
Linear Regression Algorithm

```
[8]:  from sklearn.linear_model import LinearRegression
      from sklearn.model_selection import train_test_split,cross_val_score
      from sklearn.metrics import mean_squared_error,r2_score
      from sklearn.preprocessing import OneHotEncoder
      from sklearn.preprocessing import StandardScaler
      from sklearn.linear_model import LogisticRegression
      from sklearn.neighbors import KNeighborsClassifier
      from sklearn.tree import DecisionTreeClassifier
      from sklearn.ensemble import RandomForestClassifier
      from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
      from sklearn.metrics import confusion_matrix

[9]:  # create OneHotEncoder object
      encoder = OneHotEncoder(handle_unknown='ignore')

      # transform categorical variable 'Country' into numerical
      X = encoder.fit_transform(df[['Country']])

      # assign target variable as numerical variable
      y = df['Age'].values

[10]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)

[11]: model = LinearRegression()
      model.fit(X_train, y_train)

[11]: ▾ LinearRegression
      LinearRegression()
```

```
[12]: y_pred = model.predict(X_test)

[13]: print("Training Accuracy :", model.score(X_train, y_train))
      print("Testing Accuracy :", model.score(X_test, y_test))

      Training Accuracy : 0.009281008931677603
      Testing Accuracy : -0.4699348182370586

[14]: lin_reg = LinearRegression().fit(X_train,y_train)

[15]: lin_reg.coef_[0]

[15]: -3.4348259880055174

[16]: lin_reg.intercept_

[16]: 31.301549087930532

[17]: print("{0}+{1}*Happiness Level".format(lin_reg.intercept_,lin_reg.coef_[0] ))

      31.301549087930532+-3.4348259880055174*Happiness Level

[18]: y_pred = lin_reg.predict(X_test)

[19]: y_pred[0:10]

[19]: array([27.8667231 , 31.8891976 , 32.90631004, 16.7401593 , 27.8667231 ,
             29.86275085, 32.90631004, 32.90631004, 32.90631004, 32.90631004])

[20]: np.sqrt(mean_squared_error(y_test,y_pred))

[20]: 10.198113546949791
```

Treatment with family history

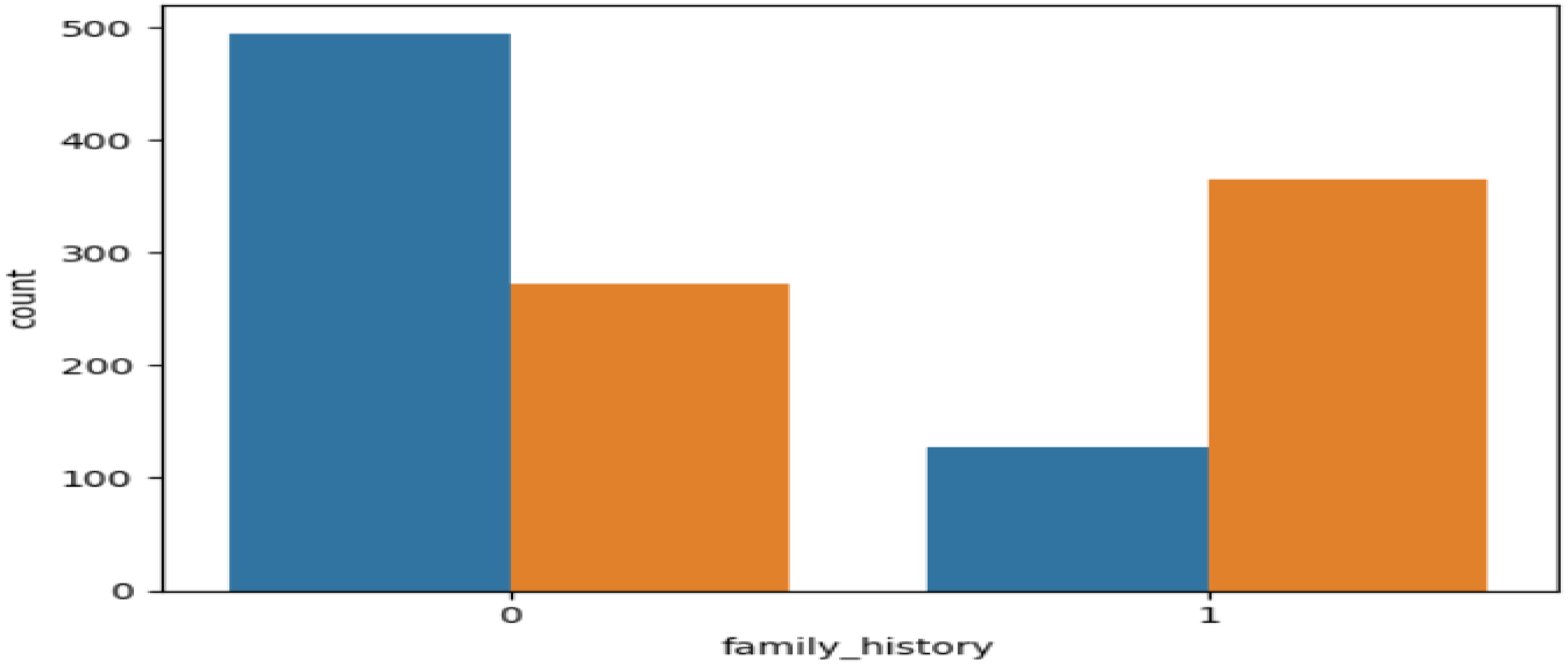
```
[21]: df["family_history"] = df["family_history"].replace({"Yes": 1, "No": 0})
df["treatment"] = df["treatment"].replace({"Yes": 1, "No": 0})
var = sns.regplot(x=df["family_history"].astype(float), y=df["treatment"].astype(float), data=df)
```



How is the treatment By family

```
[22]: sns.countplot(x="family_history", hue="treatment", data=df)
      plt.title("Treatment by Family History")
      plt.xlabel("Family History")
      plt.ylabel("Count")
      plt.show()
```

```
AttributeError: 'numpy.int64' object has no attribute 'startswith'
```



Training Accuracy and Testing Accuracy

```
[23]: scaler = StandardScaler()
      df["age_scaled"] = scaler.fit_transform(df[["Age"]])

[24]: df.drop("Age", axis=1, inplace=True)

[25]: lr_model = LogisticRegression()
      lr_model.fit(X_train, y_train)

[25]: ▾ LogisticRegression
      LogisticRegression()

[26]: y_pred = lr_model.predict(X_test)
      print("Training Accuracy :", lr_model.score(X_train, y_train))
      print("Testing Accuracy :",lr_model.score(X_test, y_test))
      lr_reg =  LogisticRegression().fit(X_train,y_train)
      lr_reg.coef_[0]
      lr_reg.intercept_
      print("{0}+{1}*Happiness Level".format(lr_reg.intercept_,lr_reg.coef_[0] ))
      df["family_history"] = df["family_history"].replace({"Yes": 1, "No": 0})
      df["treatment"] = df["treatment"].replace({"Yes": 1, "No": 0})
      var = sns.regplot(x=df["family_history"].astype(float), y=df["treatment"].astype(float),logistic=True, ci=None, data=df)
      #  # plot results
      sns.set(style="whitegrid")
      plt.title("Treatment by Family History")
      plt.xlabel("Family History")
      plt.ylabel("Treatment")
      plt.show()
```

```
Training Accuracy : 0.12258796821793416
Testing Accuracy : 0.0582010582010582
[-1.8645815  -2.17268627 -2.17268627 -2.17268627 -0.3725009   0.14533099
 -0.50912899  0.47638014  1.03927915  1.66136954  1.43032312  1.1990357
  2.43381959  2.30794448  2.19120785  2.37440162  1.92497125  2.20466527
  1.73813573  2.04846439  1.97777644  1.33513162  1.22678314  1.49808748
  1.35769631  1.14095555  0.7403819   0.21712347  0.43428462  0.81954331
 -0.4420397   0.05543766 -0.64573418 -1.61754097 -1.32239968 -0.83149755
 -0.67837563 -0.84921969 -2.17268627 -1.30042782 -1.0128323  -1.32239968
 -2.17268627 -2.17268627 -2.17268627 -1.65567533 -2.17268627 -2.17268627]+[-0.01614495 -0.00230238  0.          -0.00600279  0.          -0.00368135
 -0.0031677  -0.05041312 -0.00121391 -0.00215601 -0.00118812 -0.00233959
 -0.00114187 -0.00118929 -0.00232    -0.00980396  0.          -0.03247706
 -0.00232677 -0.00111886 -0.00901851 -0.02285126 -0.0021406  -0.00334509
 -0.00124133 -0.00111246 -0.00334692  0.          -0.02057366 -0.00733955
  0.          -0.00114187 -0.0011284  -0.00428735 -0.00226418 -0.00111246
 -0.00215601 -0.00320638 -0.00122896 -0.00483147  0.          -0.00547092
 -0.00593914 -0.00121391  0.71757744 -0.46891685 -0.00110655  0.         ]*Happiness Level
```

Logistic regression ,K Neighbour classifier, K-nearest Neighbour and Desision tree classifier

[27]:
```
y_pred = lr_model.predict(X_test)
print("Logistic Regression:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, average='macro'))
print("Recall:", recall_score(y_test, y_pred, average='macro'))
print("F1 Score:", f1_score(y_test, y_pred, average='macro'))
```

```
Logistic Regression:
Accuracy: 0.0582010582010582
Precision: 0.007271279900590247
Recall: 0.030034090909090906
F1 Score: 0.010057794375321347
C:\Users\admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\metric
ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_divisio
  _warn_prf(average, modifier, msg_start, len(result))
```

[28]:
```
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
knn= KNeighborsClassifier().fit(X_train,y_train)
```

[29]:
```
y_pred = knn.predict(X_test)
print("K-Nearest Neighbors:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, average='macro'))
print("Recall:", recall_score(y_test, y_pred, average='macro'))
print("F1 Score:", f1_score(y_test, y_pred, average='macro'))
```

```
K-Nearest Neighbors:
Accuracy: 0.06613756613756613
Precision: 0.006052216384433433
Recall: 0.040484848484848485
F1 Score: 0.010383493318275927
C:\Users\admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\metric
ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_divisio
  _warn_prf(average, modifier, msg_start, len(result))
```

[30]:
```
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
```

[30]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

Decision Tree Classifier and Random Forest Classifier

localhost:8888/notebooks/Untitled1.ipynb

```
DecisionTreeClassifier()
```

[31]:

```python
y_pred = dt.predict(X_test)
print("Decision Tree:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, average='macro'))
print("Recall:", recall_score(y_test, y_pred, average='macro'))
print("F1 Score:", f1_score(y_test, y_pred, average='macro'))
```

```
Decision Tree:
Accuracy: 0.05555555555555555
Precision: 0.031256589750237665
Recall: 0.02883801247771836
F1 Score: 0.01173526257679731
C:\Users\admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning: Precision is
ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[32]:

```python
# RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
```

[32]:

```
▾ RandomForestClassifier
RandomForestClassifier()
```

[33]:

```python
y_pred = rf.predict(X_test)
print("Random Forest:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, average='macro'))
print("Recall:", recall_score(y_test, y_pred, average='macro'))
print("F1 Score:", f1_score(y_test, y_pred, average='macro'))
```

```
Random Forest:
Accuracy: 0.05555555555555555
Precision: 0.032675021908617l7
Recall: 0.028898381229801602
F1 Score: 0.011529884983889 21
```

Test and Train Model Algorithm

jupyter Untitled1 Last Checkpoint: 12 days ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

max	1.000000	2.00000	6.157103e-01	1.000000	1.000000	1.000000	1.209466e+00	1.273620e+00	1.000000	1.000000	...	2.000000

8 rows × 24 columns

[38]:

```
from sklearn.model_selection import train_test_split

#I wanna work on 'treatment' column.
X = data.drop(columns = ['treatment'])
y = data['treatment']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

print(X_train.shape, y_train.shape)
print('-'*30)
print(X_test.shape, y_test.shape)
print('_'*30)
```

```
(937, 23) (937,)
------------------------------
(313, 23) (313,)
______________________________
```

Support Vector Classifier Algorithm

jupyter Untitled1 Last Checkpoint: 12 days ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
clf_knn.fit(X_train, y_train)

y_pred_knn = clf_knn.predict(X_test)
print('KNN accuracy :', accuracy_score(y_true=y_test, y_pred=y_pred_knn)*100)
```

```
KNN accuracy : 64.85623003194888
```

[42]:

```
steps_svc = [('Scaler', StandardScaler()),
             ('clf', SVC())]



clf_svc = Pipeline(steps=steps_svc)


clf_svc.fit(X_train, y_train)

y_pred_svc = clf_svc.predict(X_test)
print('SVC accuracy :', accuracy_score(y_true=y_test, y_pred=y_pred_svc)*100)
```

```
SVC accuracy : 70.92651757188499
```

Random Forest Classifier Algorithm

jupyter Untitled1 Last Checkpoint: 12 days ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab

Python 3 (ipykernel)

```
------------------------------
(313, 23) (313,)
______________________________
```

[39]:

```python
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier as RFC
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.tree import DecisionTreeClassifier as DT
```

[40]:

```python
steps_rfc = [('Scaler', StandardScaler()),
             ('clf', RFC(n_estimators = 40))]




clf_rfc = Pipeline(steps=steps_rfc)


clf_rfc.fit(X_train, y_train)

y_pred_rfc = clf_rfc.predict(X_test)
print('RFC accuracy: ', accuracy_score(y_true=y_test, y_pred=y_pred_rfc)*100)
```

```
RFC accuracy:  77.95527156549521
```

K Nearest Neighbour Algorithm

jupyter Untitled1 Last Checkpoint: 12 days ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab

Python 3 (ipykernel)

```
clf_rfc.fit(X_train, y_train)

y_pred_rfc = clf_rfc.predict(X_test)
print('RFC accuracy: ', accuracy_score(y_true=y_test, y_pred=y_pred_rfc)*100)
```

```
RFC accuracy:  77.95527156549521
```

[41]:

```
steps_knn = [('Scaler', StandardScaler()),
             ('clf', KNN(n_neighbors = 5))]



clf_knn = Pipeline(steps=steps_knn)


clf_knn.fit(X_train, y_train)

y_pred_knn = clf_knn.predict(X_test)
print('KNN accuracy :', accuracy_score(y_true=y_test, y_pred=y_pred_knn)*100)
```

```
KNN accuracy : 64.85623003194888
```

Decision Tree Algorithm

```
y_pred_svc = clf_svc.predict(X_test)
print('SVC accuracy :', accuracy_score(y_true=y_test, y_pred=y_pred_svc)*100)
```

```
SVC accuracy : 70.92651757188499
```

[44]:
```
steps_dt = [('Scaler', StandardScaler()),
            ('clf', DT())]



clf_dt = Pipeline(steps=steps_dt)


clf_dt.fit(X_train, y_train)

y_pred_dt = clf_dt.predict(X_test)
print('DT accuracy :', accuracy_score(y_true=y_test, y_pred=y_pred_dt)*100)
```

```
DT accuracy : 73.48242811501598
```

[ ]:

Conclusion

In conclusion, through this Public Health Awareness Campaign, we have covered a range of important topics, from Jupyter Notebook code implementation and data extraction to the utilization of various machine learning algorithms. By leveraging these technologies, we are confident in our ability to raise awareness effectively and make a positive impact on public health. Together, let's work towards a healthier future!