Specific words can help Predict Internet Sales

Problem Definition:

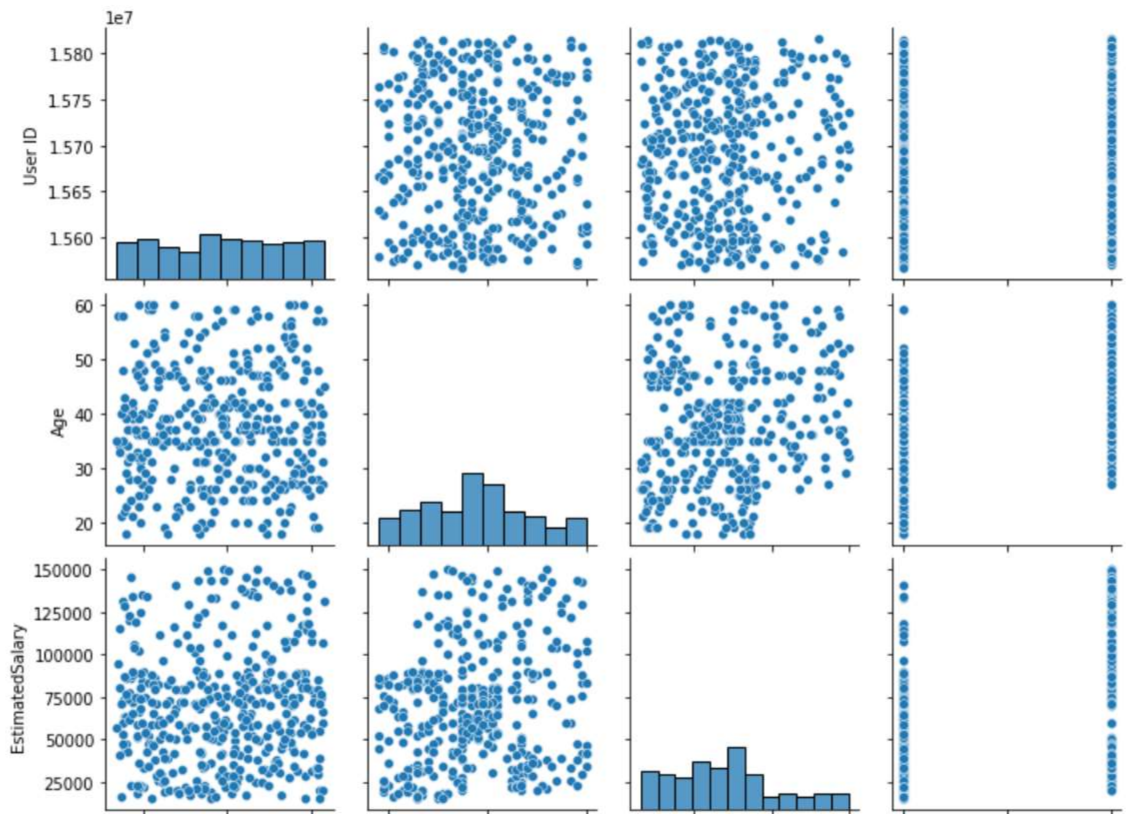
This project aims to make accurate purchase predictions based on existing known words used in advertising so that marketing methods could reveal top-selling words for sales. Used Machine-learning algorithms to create models that will predict sales. The data for this model consists of Google AdWords. Twenty-five percent of this training dataset was split into a test dataset with corresponding purchases to determine the accuracy and error of the model.

Data Wrangling

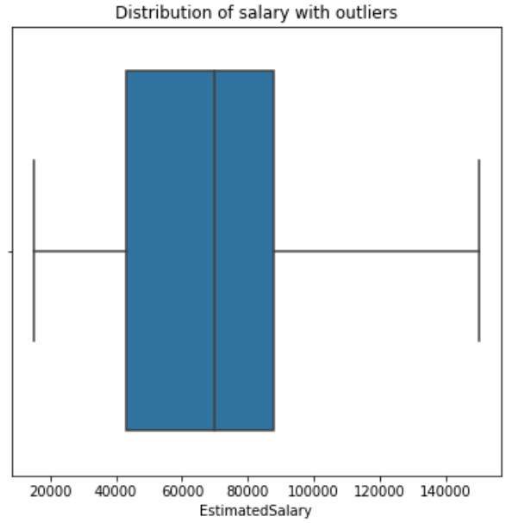
```
df.describe()
```

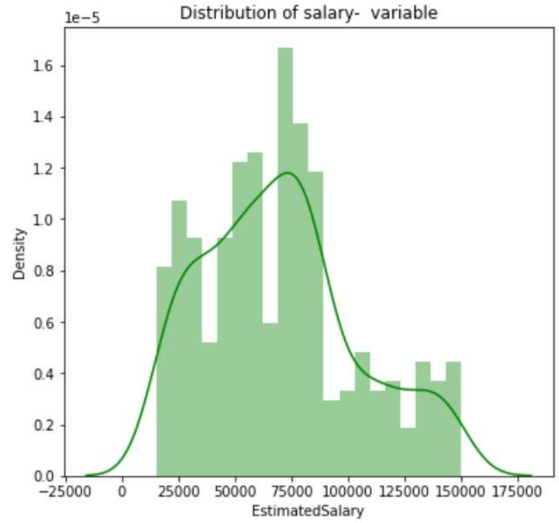
	User ID	Age	EstimatedSalary	Purchased
count	4.000000e+02	400.000000	400.000000	400.000000
mean	1.569154e+07	37.655000	69742.500000	0.357500
std	7.165832e+04	10.482877	34096.960282	0.479864
min	1.556669e+07	18.000000	15000.000000	0.000000
25%	1.562676e+07	29.750000	43000.000000	0.000000
50%	1.569434e+07	37.000000	70000.000000	0.000000
75%	1.575036e+07	46.000000	88000.000000	1.000000
max	1.581524e+07	60.000000	150000.000000	1.000000

```
sns.pairplot(df)
# to show
plt.show()
```



Explanatory Data Analysis



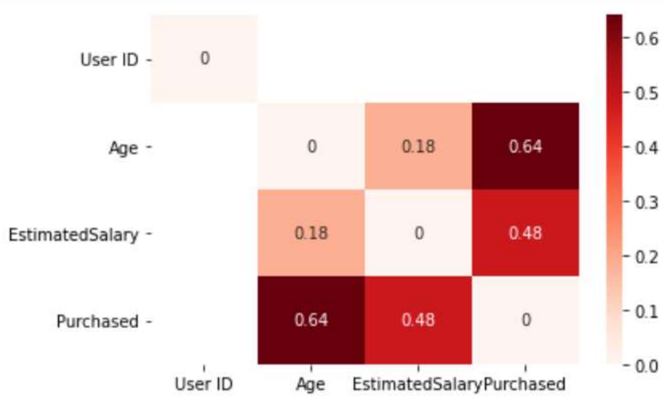


NOTE:

There are no outliers with a somewhat distributed variable

Exploring relationship between the feature variables and the target

```
sns.heatmap(corr, annot=True, cmap=plt.cm.Reds)
plt.show()
```



Algorithms and Machine Learning

```
Load Machine Learning Libraries¶
Apply the following 4 ML models

* Logistics Regression
* Decision Tree
* Random Forrest
* KNN
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
```

Split the X and y into 75/25 training and testing data subsets

```
#train test split

X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.25, random_state=42)
```

Feature Scaling For Standardization

```
sc_X=StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

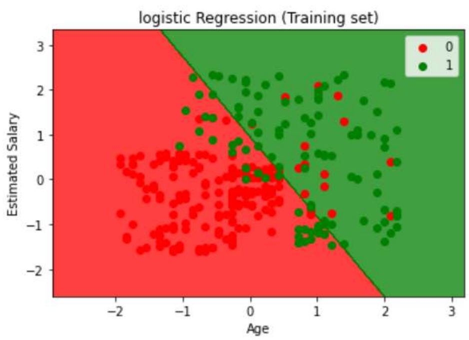
NOTE:

Using Scaling to improve model performance

Visualizing Training test results

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
            alpha = 0.75, cmap = ListedColormap(('red','green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

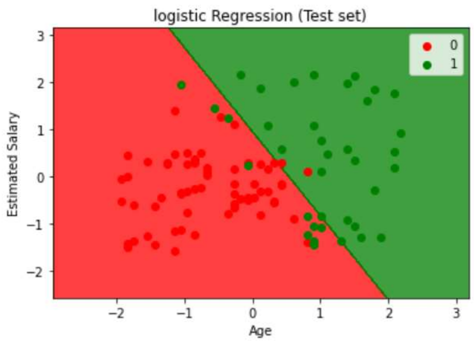
```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence
in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row
if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence
in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row
if you intend to specify the same RGB or RGBA value for all points.
```



There are a few mis-classifications with internet purchases.

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red','green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('logistic Regression (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence
in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row
if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence
in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row
if you intend to specify the same RGB or RGBA value for all points.
```



Confusion Matrix Evaluation

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test, y_pred)
cm
```

```
11]: array([[61,  2],
            [12, 25]], dtype=int64)
```

NOTE: Confusion Matrix Break down

```
N = 100

TN & FN is the Predicted NO

FP & TP is the Predicted YES

Accuracy:
    (TN+TP)/N
    (61 + 25)/100 = 86%

Misclassification Rate:
    (FP +FN)/N
    (2 + 12)/100 = 14%
```

Training a Random Forrest

```
[ ]: rfc = RandomForestClassifier(n_estimators=100)
     rfc.fit(X_train, y_train)
```

```
113]: RandomForestClassifier()
```

```
[ ]: rfc_pred = rfc.predict(X_test)
```

```
[ ]: print(confusion_matrix(y_test, rfc_pred))
```

```
[[57  6]
 [ 4 33]]
```

```
[ ]: from sklearn.metrics import classification_report
     print(classification_report(y_test, rfc_pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.90      0.92        63
           1       0.85      0.89      0.87        37

    accuracy                           0.90       100
   macro avg       0.89      0.90      0.89       100
weighted avg       0.90      0.90      0.90       100
```

NOTE: Confusion Matrix Break down

```
N = 100

TN & FN is the Predicted NO

FP & TP is the Predicted YES

Accuracy:
    (TN+TP)/N
    (57 + 33)/100 = 90%

Misclassification Rate:
    (FP +FN)/N
    (6 + 4)/100 = 10%
```

Training a Decision Tree

```
dtree = DecisionTreeClassifier()
dtree.fit(X_train, y_train)
```

```
]: DecisionTreeClassifier()
```

```
dtree_pred = dtree.predict(X_test)
dtree_pred
```

```
]: array([1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0,
          1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
          0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1,
          1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0,
          0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0], dtype=int64)
```

```
print(confusion_matrix(y_test, dtree_pred))
```

```
[[56  7]
 [ 9 28]]
```

```
print(classification_report(y_test, dtree_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.89      0.88        63
           1       0.80      0.76      0.78        37

    accuracy                           0.84       100
   macro avg       0.83      0.82      0.83       100
weighted avg       0.84      0.84      0.84       100
```

NOTE: Confusion Matrix Break down

N = 100

TN & FN is the Predicted NO

FP & TP is the Predicted YES

Accuracy: (TN+TP)/N (56 + 28)/100 = 84%

Misclassification Rate: (FP +FN)/N (9 + 7)/100 = 16%

Training a K Nearest Neighbors

```
knn = KNeighborsClassifier(n_neighbors = 6, metric = 'minkowski', p =2)
knn.fit(X_train, y_train)
```

```
[124]: KNeighborsClassifier(n_neighbors=6)
```

```
knn_pred = knn.predict(X_test)
knn_pred
```

```
[126]: array([1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1,
       1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0], dtype=int64)
```

```
print(confusion_matrix(y_test, knn_pred))
```

```
[[60  3]
 [ 5 32]]
```

```
print(classification_report(y_test, knn_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.95      0.94        63
           1       0.91      0.86      0.89        37

    accuracy                           0.92       100
   macro avg       0.92      0.91      0.91       100
weighted avg       0.92      0.92      0.92       100
```

NOTE: Confusion Matrix Break down

N = 100

TN & FN is the Predicted NO

FP & TP is the Predicted YES

Accuracy: (TN+TP)/N (60 + 32)/100 = 92%

Misclassification Rate: (FP +FN)/N (5 + 3)/100 = 8%

In Conclusion:

I decided to do different classification techniques to predict internet sales. The purpose was to find the best performers: logistic Regression, Decision Trees, Random Forrest support vectors to get it done. Engineering offered the data in 2 different warehouses, Hadoop, and Apache Spark. I worked with the management and Analyst team to understand what information was collected from Google AdWords and Facebook before extracting data. After running several Algorithms, Both Random Forrest and K nearest Neighbor provided an accuracy of 92%.