FUTURE SALES PREDICTION

PHASE 4: DEVELOPMENT PART 2

INTRODUCTION

Future sales prediction, often referred to as sales forecasting, is a crucial aspect of business strategy. It involves the use of statistical models and predictive analytics to estimate a company's future sales performance. By examining historical sales data, market trends, and other relevant factors, businesses can make informed decisions regarding inventory management, resource allocation, and overall growth strategies. Sales predictions enable companies to adapt to changing market conditions, optimize marketing efforts, and ensure sustainable success in a competitive business landscape.

WORKS DONE IN PREVIOUS PHASES

DEFINITION PHASE

In this phase we defined the problem that develop a predictive model that uses historical sales data to forecast future sales for a retail company. The objective is to create a tool that enables the company to optimize inventory management and make informed business decisions based on data driven sales predictions. These involves data preprocessing, feature engineering, model selection, training, and evaluation.

INNOVATION PHASE

In the innovation phase of our future sales prediction project, you can explore advanced techniques and methods to improve the accuracy of your sales forecasting model like Prophet Forecasting model and LSTM

DEVELOPMENT PHASE 1

In this phase we loaded the dataset which is provided for us and pre-processed the data by using python library packages and necessary methods to implement it

Provided dataset for us

https://www.kaggle.com/datasets/chakradharmattapalli/future-salesprediction

PHASE 4

- In this phase we are going to test the model which we are pre-processed by using some of the models and going to evolve those models

- This can be executed by using

    1. Feature engineering
    2. Model testing
    3. Evaluation

Feature engineering

We can Create additional features that could enhance the predictive power of the model, such as time-based features

- day of the week
- month

TRAIN_TEST_SPLIT

Sklearn

- Scikit-learn, often abbreviated as sklearn, is a popular machine learning library in Python.

- It provides a wide range of tools and algorithms for tasks related to data analysis, machine learning, and data mining.

- Scikit-learn is open-source and built on top of other Python libraries such as NumPy, SciPy, and Matplotlib.

Key features of sklearn

- Supervised Learning
- Unsupervised Learning
- Preprocessing
- Model Selection
- Model Evaluation
- Pipelines
- Feature Extraction
- Datasets

Code we used in our project:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

Here x_train,x_test,y_train,y_test are used for training and testing dataset data

```
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

Some of the features we used in this project are

- Mean_squared_error
- r2_score
- cross_val_score
- GridSearchCV
- KNeighborsRegressor
- SVR
- DecisionTreeRegressor
- RandomForestRegressor

KNN

- K-Nearest Neighbors (KNN) is a simple and intuitive machine learning algorithm used for classification and regression tasks.

- It is a type of instance-based or lazy learning algorithm, meaning it does not build an explicit model during training but instead memorizes the training data.

- KNN makes predictions by finding the K training examples (neighbors) in the dataset that are closest to a given input data point and then using a majority vote or weighted average of their target values to make predictions.

The code for our project using knn:

```
knn=KNeighborsRegressor().fit(x_train,y_train)
```

```
knn
```

Output:

```
KNeighborsRegressor
KNeighborsRegressor()
```

Code for train test split:

```
knn_train_predict=knn.predict(x_train)
knn_test_predict=knn.predict(x_test)
```

```
print(knn_train_predict,knn_test_predict)
```

Output:

```
[19.72  5.96  8.74  9.38 13.48 14.66  9.98 19.32 19.66 18.14 17.28  5.96
 14.5   7.6  10.42 12.52 10.7  13.36 17.28 12.52 18.72 16.04 15.76 15.76
 13.5   4.44  7.9  13.48 19.72 16.52 19.32 19.5  21.86 12.52 12.1  14.72
 12.1   5.96  8.2  15.28 16.12  8.58 21.56 13.5   8.2  10.54 16.02 10.86
 12.1  11.52 19.32 12.84 13.42 11.1  18.08 14.66 14.   14.66 18.72 16.66
 21.56 18.08  8.66 11.44 15.06 18.32  8.74 18.72 16.12  8.4  16.12 13.64
 13.44  7.9   9.72 14.6   9.98 15.28 13.78 17.28 13.48 21.56 14.74 19.66
 16.02  8.58 10.76  9.66 13.8  19.74 12.16 16.7  20.14 18.72 18.14 10.88
  8.58 19.12  5.96  5.26 11.1   9.72 13.5  10.42 13.42  8.54 13.6  11.52
 16.94 14.06  9.72 10.18 10.2  14.66 13.5  14.46 12.1  12.52  4.44 12.16
 14.   15.76 19.66 19.74 10.88 16.12 16.02 20.54 19.72 19.32 13.04  5.96
 17.22 16.52 19.74 19.48 14.06 16.66  8.38 11.1 ] [13.64 12.58 21.56 11.94  7.9  16.94 17.72 15.28
12.16 11.58 19.66 13.9
 19.12 17.72  8.74 14.9  18.32 21.86 12.52 21.86 19.66 19.32  7.9  13.44
 19.66 19.48  8.74 13.64 21.56 19.48 21.26 18.08 19.32 17.28 15.38 11.8
 11.52 19.12 16.94 21.56 17.28 15.76 14.9   8.38  8.58 10.94 10.54 18.08
 10.7  10.76 12.52 13.42 13.36 18.72 13.24 15.38 18.72 13.5  17.28 10.2 ]
```

The output implies that the predicted trained and tested values according to the dataset.

```
Result=pd.DataFrame(columns=['Model','Train R2','Test R2','Test
RMSE','Variance'])
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score


r2 = r2_score(y_test, knn_test_predict)
r2_train = r2_score(y_train, knn_train_predict)
```

```
rmse = np.sqrt(mean_squared_error(y_test, knn_test_predict))
variance = r2_train - r2

Result = Result.append({
    "Model": "K-Nearest Neighbors",
    "Train R2": r2_train,
    "Test R2": r2,
    "Test RMSE": rmse,
    "Variance": variance
}, ignore_index=True)

print("R2:", r2)
print("RMSE:", rmse)
```

Output:

```
       S.no         TV           Radio        Newspaper    sales
count  200.000000   200.000000   200.000000   200.000000   200.000000
mean   99.567000    145.949000   23.288000    30.577500    14.022500
std    57.771081    86.157663    14.826852    21.757446    5.217457
min    1.000000     0.700000     0.000000     0.300000     1.600000
25%    49.750000    72.700000    10.075000    12.875000    10.375000
50%    99.500000    148.500000   22.900000    25.750000    12.900000
75%    149.250000   218.425000   36.525000    45.100000    17.400000
max    199.000000   296.400000   49.600000    114.000000   27.000000
```

The output implies the mean squared error and r2_score

Mean Squared Error

- Mean Squared Error (MSE) is a common metric used to measure the performance or accuracy of a regression model.
- It quantifies how well the predictions made by a regression model align with the actual values in the dataset.
- MSE measures the average of the squared differences between the predicted values and the actual values.
- Lower MSE values indicate a better fit of the model to the data.

The formula for calculating the Mean Squared Error (MSE) is as follows:

MSE = (1/n) * $\Sigma(y_i - \hat{y}_i)^2$

Where

MSE: Mean Squared Error

n: the number of data points

y_i: the actual target value for the i-th data point

$\hat{y}_i$: the predicted target value for the i-th data point

Σ: the summation symbol, which means to sum over all data points

R2_score:

- R-squared (R2) score, also known as the coefficient of determination, is a statistical metric used to evaluate the goodness of fit of a regression model.

- In simpler terms, it tells you how well the model predicts the variance in the data.

- R2 values range between 0 and 1

R2 = 0 indicates target variable, meaning it is a poor fit for the data.

R2 = 1 indicates that the model perfectly explains the variance in the target variable, making it an ideal fit for the data.

Formula:

R2 = 1 - (SSR / SST)

Where:

SSR (Sum of Squared Residuals) is the sum of the squared differences between the predicted values and the actual values.

SST (Total Sum of Squares) is the sum of the squared differences between the actual values and the mean of the actual values.

```
Result.head()
```

Output:

	Model	Train R2	Test R2	RMSE	Variance
0	K-Nearest Neighbors	0.658333	0.575513	3.504574	0.082820

Model Testing

Support Vector Regression

- Support Vector Regression (SVR) is a variation of the Support Vector Machine (SVM) algorithm that is used for regression tasks.
- While traditional SVMs are primarily used for classification, SVR is specifically designed for regression problems.
- SVR is a powerful and versatile regression technique that works by finding a hyperplane (or hyperplanes) that best fits the data while minimizing the margin violations.

Code for our project using SVR

```
svr=SVR().fit(x_train,y_train)
svr
```

Output:

```
SVR
SVR()
```

```
SVR()
```

Output:

```
SVR
SVR()
```

```
svr_train_predict=svr.predict(x_train)
svr_test_predict=svr.predict(x_test)
print(svr_train_predict,svr_test_predict)
```

Output:

```
[17.37929494  7.76237612  8.16667884 10.12194867 12.4454335  15.13241051
 12.12008317 17.64610057 17.6039259  17.94187708 18.18399744  7.81301564
 13.08066667  7.93154723  9.37863687 11.16271395  9.69749316 14.8602086
 18.1619553  11.1250958  18.13969555 16.69976218 14.34537647 14.46126778
 13.24606313  7.65844601  8.29163402 12.4454335  17.44380537 16.94682756
 17.6706073  17.74064179 17.57482687 11.09651529 11.43094188 14.52873663
 11.43917779  7.80929743  8.69797871 17.09121421 16.50049757  8.60047601
 17.32845287 13.28565846  8.73872856  9.47242569 18.13101319 11.81406886
 11.42267068 11.47580366 17.47581681 12.32604895 12.68815867 12.10695912
 17.87810629 15.01654564 14.62002263 15.13241051 18.13644669 16.79114905
 17.19467333 17.88757676  8.97725147 11.95960135 13.98611446 16.41068888
  8.21580405 18.16104878 16.48858441  8.8034119  16.54203639 13.66333051
 12.6160006   8.26100174 10.48609609 13.14272496 12.14598005 17.14827027
 12.51614864 18.15741339 12.46592813 17.16471972 13.10974436 17.5956702
 18.11644168  8.52743862 11.62269124 10.01879351 12.93181487 18.23750861
 11.89647331 18.14955689 18.23640649 18.08783922 17.92426013 12.28138254
  8.62892834 16.01180211  7.78010958  7.69864109 12.05867323 10.55953899
 13.25676952  9.31000205 12.69552799  8.00446932 13.56008941 11.47176926
 15.8055559  12.99976217 10.33013493 12.15366146  9.79706195 15.05299962
 13.32246892 12.8289351  11.40602224 11.22719122  7.56693544 11.93888362
 14.72442042 14.33448944 16.27722783 18.23234948 12.22603791 16.54203639
 18.11644168 18.22340967 17.42558639 17.68786921 13.20045927  7.80190727
 18.05600709 16.94136752 18.23959145 17.28361927 12.9817203  16.84180344
  7.99131063 12.06683836] [13.66773438 13.38300422 17.41313539 11.01384637  8.25593484
15.81182319
 17.80274639 17.15340388 11.86220351 10.82026447 17.67998834 12.39066882
...
 11.47580366 15.84314837 17.89232764 17.04272313 18.1619553  14.40020798
 15.36729577  7.99568325  8.59480898 10.67570818  9.39113185 17.80504854
  9.691256   11.54693152 11.153362   12.70788015 14.78904497 18.06135504
 12.76079444 15.62366649 18.09559065 13.24962427 18.20132193  9.78464231]
```

```python
r2 = r2_score(y_test, svr_test_predict)
r2_train = r2_score(y_train, svr_train_predict)
rmse = np.sqrt(mean_squared_error(y_test, svr_test_predict))
variance = r2_train - r2

Result = Result.append({
    "Model": "Support Vector Machine",
    "Train R2": r2_train,
    "Test R2": r2,
    "Test RMSE": rmse,
    "Variance": variance
}, ignore_index=True)

print("R2:", r2)
print("RMSE:", rmse)
```

Output:

```
R2: 0.5726393654018211
RMSE: 3.516415309123736
```

```
Result.head()
```

Output:

```
           Model                          Train R2       Test R2   RMSE     Variance
0          K-Nearest Neighbors               0.658333 0.575513 3.504574 0.082820
1          Support Vector Machine            0.566175 0.572639 3.516415 -0.00646
```

Evaluation

Statsmodel:

- StatsModels is a Python library for estimating and interpreting various statistical models.
- It is primarily used for statistical analysis, hypothesis testing, and exploring relationships within data.
- StatsModels provides a wide range of statistical models and tools for various purposes, such as

1. linear regression
2. logistic regression
3. time series analysis
4. hypothesis testing

Code for statsmodel in our project:

```
import statsmodels.api as sm
x_train_constant=sm.add_constant(x_train)
model=sm.OLS(y_train,x_train_constant).fit()
model.params
```

Output:

```
const    7.225998
x1       0.047035
dtype: float64
```

```
model.summary()
```

Output:

```
OLS Regression Results
Dep. Variable:   Sales   R-squared:       0.575
Model:  OLS     Adj. R-squared: 0.572
Method:          Least Squares  F-statistic:       186.9
Date:   Tues, 24 oct 2023        Prob (F-statistic):          1.96e-27
Time:   13:13:52         Log-Likelihood: -367.12
No. Observations:        140     AIC:     738.2
Df Residuals:    138     BIC:     744.1
Df Model:        1
Covariance Type:         nonrobust
coef    std err  t       P>|t|   [0.025  0.975]
const   7.1564  0.563   12.722  0.000   6.044   8.269
x1      0.0456  0.003   13.670  0.000   0.039   0.052
Omnibus:         0.224   Durbin-Watson: 2.154
Prob(Omnibus): 0.894     Jarque-Bera (JB):        0.393
Skew:   0.028   Prob(JB):        0.821
Kurtosis:        2.746   Cond. No.        335.
```

Analysis of the output:

Dependent Variable (Dep. Variable):

This is the variable we predicted which in this case is "Sales."

R-squared (R-squared):

R-squared measures the proportion of the variance in the dependent variable that is explained by the independent variable(s). An R-squared of 0.575 means that approximately 57.5% of the variation in sales is explained by the independent variable x1.

Model:

This section provides information about the type of model used (OLS, Ordinary Least Squares).

Adjusted R-squared (Adj. R-squared):

This is a modified version of R-squared that adjusts for the number of independent variables in the model. It is 0.572, indicating a slightly lower R-squared value after accounting for the model's complexity.

Method:

The method used for this analysis is "Least Squares," which is the standard method for estimating the coefficients in a linear regression model.

F-statistic and Prob (F-statistic):

The F-statistic is used to test the overall significance of the model. A high F-statistic (186.9 in this case) suggests that the model as a whole is statistically significant. The very low p-value (1.96e-27) indicates that the model is highly significant.

Date and Time:

These details indicate when the analysis was conducted.

Log-Likelihood:

The log-likelihood is a measure of how well the model fits the data. In this case, it is -367.12.

No. Observations:

This tells you the number of data points in your dataset, which is 140 in this case.

AIC and BIC:

AIC (Akaike Information Criterion) and BIC (Bayesian Information Criterion) are measures of the model's goodness of fit. Lower values suggest a better fit. In this output, AIC is 738.2, and BIC is 744.1.

Degrees of Freedom (Df Residuals, Df Model):

Df Residuals is the degrees of freedom for the residuals, and Df Model is the degrees of freedom for the model. In this case, there are 138 degrees of freedom for the residuals and 1 degree of freedom for the model.

Covariance Type:

This mentions that the covariance type used is "nonrobust," indicating that the standard OLS method without robust standard errors was applied.

Coefficients:

This provides information about the coefficients of the model. You have two coefficients here:

- "const" represents the intercept or constant term, which is 7.1564. It indicates the expected value of Sales when x1 is 0.

- "x1" represents the coefficient of the independent variable x1, which is 0.0456. It indicates the change in Sales for a one-unit change in x1.

The columns std err, t, P>|t|, [0.025, 0.975] provide additional information about the coefficients.

Omnibus, Durbin-Watson, Prob(Omnibus), Jarque-Bera (JB), Skew, Kurtosis:

These are various statistical tests and measures to assess the assumptions and quality of the model residuals. These values can be used to check for issues like normality and autocorrelation.

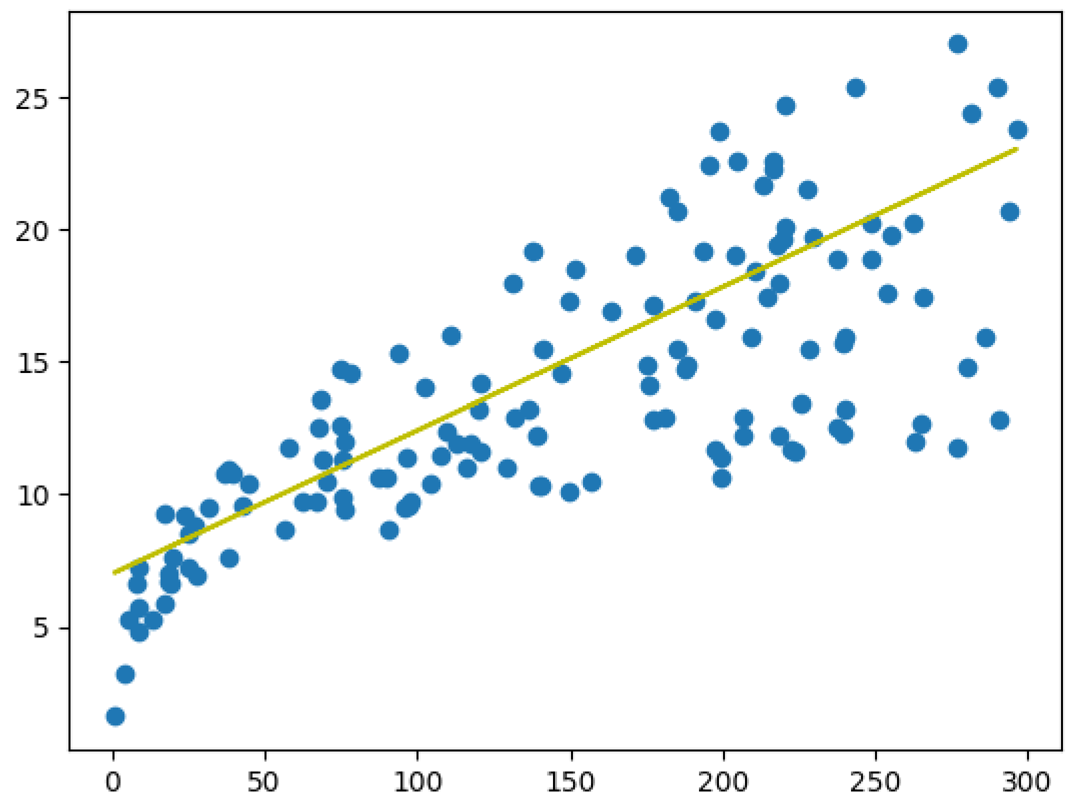
Cond. No.:

This is the condition number, which can indicate multicollinearity in the model. A high condition number suggests potential multicollinearity.

This output is the result of a linear regression analysis showing the relationship between Sales and x1, with information about the model's goodness of fit, significance, and various statistics related to the coefficients and residuals.

```
plt.scatter(x_train,y_train)
plt.plot(x_train,6.9955+0.0541*x_train,'y')
plt.show()
```

Output:



Conclusion:

The ultimate goal is to deliver a valuable forecasting tool that enhances our company's growth and profitability, like putting a masterpiece on display to benefit our business and the world around us. This project is a canvas, and data is our palette; let's create something truly remarkable.We delivered it in a good and feasible way.